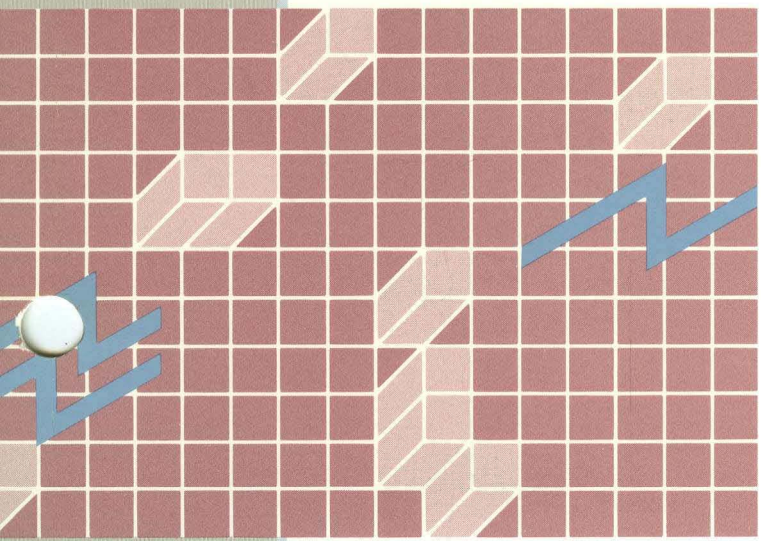
High Performance LEVEL II COBOL™

Language Reference



High Performance LEVEL II COBOL™

Language Reference Manual

LEVEL II COBOL LANGUAGE REFERENCE MANUAL

AMENDMENT RECORD

Issue Number	Dated	Inserted by	Signature	Date
2	October 1982	- Incorporated in this reprint -		
3	December 1982	- Addendum incorporated in this reprint -		
4	February 1983	- Addendum 2 incorporated in this reprint -		
5	July 1983	- Incorporated in this reprint -		
6	February 1984	- Typographical corrections -		
7	April 1984	- Addendum 3 incorporated in this reprint -		
8	May 1985	-Incorporated in this reprint -		

PREFACE

This manual describes the LEVEL II COBOL language for programming microcomputers. LEVEL II COBOL is based on the ANSI COBOL standard X3.23 (1974) (see Acknowledgement). It also describes the additional LEVEL II COBOL features that exploit the capabilities of microprocessors.

Each release of LEVEL II COBOL is characterized by a two-digit code in the form of *v.r* where *v* is the version number and *r* is the release number within the version.

AUDIENCE

This manual is intended for programmers already familiar with COBOL on other equipment.

MANUAL ORGANIZATION

Chapters 1 through 4 of the manual apply to all users and describe basic features of the language. Chapters 5 through 7 describe language features for programming the three file organization formats supported: sequential, relative and indexed.

Chapters 8 through 14 apply to all users and describe additional features and facilities available with the standard language. The appendices supply reference information pertinent to all systems.

*

RELATED PUBLICATIONS

No discussion of operating the LEVEL II COBOL compiler or Run-Time System is incorporated in this manual. Please refer to the document:

LEVEL II COBOL Operating Guide
(for use with the relevant Operating System)

In the general formats, the ellipsis represents the position at which repetition may occur at the user's option. The portion of the format that may be repeated is determined as follows:

Given ... in a clause or statement format, scanning right to left, determine the } or] immediately to the left of the ...; continue scanning right to left and determine the logically matching { or [; the ... applies to the words between the determined pair of delimiters.

9. The term identifier means either a data-name or a subscripted data-name. An identifier takes the following form:

```
data-name-1  [(  {data-name-2 }  )]
             [   {literal-1   }   ]
```

data-name-2 or literal-1 must be a positive integer in the range 1 to the number of elements in the table.

10. All numbers in this manual are decimal unless otherwise specified. The letters hex after a number indicate that the number is in hexadecimal format.

11. Italics are used when a word or concept is first introduced.

Headings are presented in this manual in the following order of importance:

CHAPTER *N*
TITLE — Chapter Heading

ORDER ONE HEADING
Order Two Heading
ORDER THREE HEADING
Order Four Heading — Text two lines down

Order Five Heading: — Text on same line

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

CHAPTER 2
COBOL CONCEPTS

CHAPTER 4
TABLE HANDLING

CHAPTER 6
RELATIVE INPUT AND OUTPUT

CHAPTER 7
INDEXED INPUT AND OUTPUT

CHAPTER 10
LIBRARY

CHAPTER 11
DEBUG AND INTERACTIVE DEBUGGING

CHAPTER 13
COMMUNICATION

FIGURES

CHAPTER 1

INTRODUCTION

WHAT IS LEVEL II COBOL?

COBOL (COmmon Business Oriented Language) is the most widely and extensively used language for the programming of commercial and administrative data processing.

LEVEL II COBOL is a compact, interactive and standard COBOL Language System which is designed for use on microprocessor-based computers and intelligent terminals.

It is based on the ANSI COBOL as specified in **American National Standard Programming Language COBOL** (ANSI X3.23 1974). The following modules are fully implemented at Level II:

- Nucleus
- Table Handling
- Sequential Input and Output
- Relative Input and Output
- Indexed Input and Output
- Sort-Merge
- Segmentation
- Library
- Inter-Program Communication
- Debug
- Communications

This manual is intended as a reference work for LEVEL II COBOL programmers and material from the ANSI COBOL language standard document is included.

The package has been proved to meet and exceed the COBOL ANSI standard X3.23 and has been certified by the Federal Software Testing Center (FSTC) under the direction of the General Services Administration (GSA) as validated at Federal High Level.

LEVEL II COBOL programs are created using a conventional text editor. The compiler compiles the programs and the Run-Time System links with the compiled output to form a running user program. A listing of the LEVEL II COBOL program is provided by the compiler during compilation. Error messages are inserted in the listing.

LEVEL II COBOL is designed to be interfaced easily to any microprocessor operating system. Detailed operating characteristics are dependent on the particular host operating system used and are defined in the appropriate **LEVEL II COBOL Operating Guide.**

*

FORMATS AND RULES

General Format

A general format is the specific arrangement of the elements of a clause or a statement. Throughout this document a format is shown adjacent to information defining the clause or statement. When more than one specific arrangement is permitted, the general format is separated into numbered formats. Clauses must be written in the sequence given in the general formats. (Clauses that are optional must appear in the sequence shown if they are used). In certain cases, stated explicitly in the rules associated with a given format, the clauses may appear in sequences other than that shown. Applications, requirements or restrictions are shown as rules.

Syntax Rules

Syntax rules are those rules that define or clarify the order in which words or elements are arranged to form larger elements such as phrases, clauses, or statements. Syntax rules also impose restrictions on individual words or elements.

These rules are used to define or clarify how the statement must be written, that is, the order of the elements of the statement and restrictions on what each element may represent.

Indicator Area

An asterisk * in this area marks the line as documentary comment only. Such a comment line can appear anywhere in the program after the Identification Division header. Any characters from the ASCII character set can be included in Area A and Area B of the line.

A stroke /, in the indicator area acts as a comment line above but causes the page to eject before printing the comment.

A "D" in the indicator area represents a debugging line. Areas A and B may contain any valid COBOL sentence.

A "-" in the indicator area represents a continuation of the previous line without spaces or the continuation of a non-numeric literal (see Chapter 2).

```
000330      02  CRT-PROD-DESC PIC X(24).
000340      02  FILLER PIC X(56).
000350      02  CRT-UNIT-SIZE PIC 9(4).
000360      02  FILLER PIC X.
000370 PROCEDURE DIVISION.
000380 SR1.
000390      DISPLAY SPACE.
000400      OPEN I-O STOCK-FILE.
000410      DISPLAY SCREEN-HEADINGS.
000420 NORMAL-INPUT.
000430      MOVE SPACE TO ENTER-IT.
000440      DISPLAY ENTER-IT.
000450 CORRECT-ERROR.
000460      ACCEPT ENTER-IT.
000470      IF CRT-STOCK-CODE = SPACE GO TO END-IT.
000480      IF CRT-UNIT-SIZE NOT NUMERIC GO TO CORRECT-ERROR.
000490      MOVE CRT-PROD-DESC TO PRODUCT-DESC.
000500      MOVE CRT-UNIT-SIZE TO UNIT-SIZE.
000510      MOVE CRT-STOCK-CODE TO STOCK-CODE.
000520      WRITE STOCK-ITEM; INVALID GO TO CORRECT-ERROR.
000530      GO TO NORMAL-INPUT.
000540 END-IT.
000550      CLOSE STOCK-FILE.
000560      DISPLAY SPACE.
000570      DISPLAY "END OF PROGRAM".
000580      STOP RUN.

* LEVEL II COBOL  V2.5                stock1.cbl                  Page 0002

*                                  29-Jan-85  10:41
* LEVEL II COBOL  V2.5    revision 2                    URN 0000/AA00/00000A
* Compiler  Copyright (c) 1982 Micro Focus Ltd          REF CNB-100100022
*
* Errors=00000 Data=01024 Code=00512 Dict=00354:26159/26513 FIPS flagging off
```

Cols 12-72
Area B

Col 7
Indicator
Area

Cols.
1-6
Sequence
Number

Cols 8-11
Area A

Figure 1-1. Sample Program Listing Showing Source Format (Cont)

CHAPTER 2

COBOL CONCEPTS

LANGUAGE CONCEPTS

Character Set

The most basic and indivisible unit of the language is the character. The set of characters used to form LEVEL II COBOL character-strings and separators includes the letters of the alphabet, digits and special characters. The character set consists of the characters defined below:

0 to 9
A to Z
a to z (Reserved and User-defined Word Characters read as: A to Z)
Space

+	Plus sign
-	Minus sign or hyphen
*	Asterisk
/	Oblique Stroke/Slash
=	Equal sign
$	Dollar sign
.	Full stop or decimal point
,	Comma or decimal point
;	Semicolon
"	Quotation mark
(	Left Parenthesis
)	Right Parenthesis
>	Greater than symbol
<	Less than symbol

The LEVEL II COBOL language is restricted to the above character set, but the content of non-numeric literals, comment lines and data may include any of the characters from the ASCII character set. See the **LEVEL II COBOL Pocket Guide**

5. Pseudo-text delimiters are separators. An opening pseudo-text delimiter may be immediately preceded by a space; a closing pseudo-text delimiter must be immediately followed by one of the separators space, comma, semicolon, or period.

   Pseudo-text delimiters may appear only in balanced pairs delimiting pseudo-text. (See Chapter 10 **Library**)

6. The separator space may optionally immediately precede all separators except the following:

   a. As specified by reference format rules see **Reference Format** in this Chapter.

   b. The separator closing quotation mark. In this case, a preceding space is considered as part of the nonnumeric literal and not as a separator.

   c. The opening pseudo-text delimiter, where the preceding space is required.

7. The separator space is optional and can immediately follow any separator except the opening quotation mark. In this case, a following space is considered as part of the nonnumeric literal and not as a separator.

Any punctuation character which appears as part of the specification of a PICTURE character-string (see Chapter 3) or numeric literal is not considered as a punctuation character, but rather as a symbol used in the specification of that PICTURE character-string or numeric literal. PICTURE character-strings are delimited only by the separators space, comma, semicolon, or period.

The rules established for the formation of separators do not apply to the characters which comprise the contents of nonnumeric literals, comment-entries, or comment lines.

CHARACTER STRINGS

A character-string is a character or a sequence of contiguous characters which forms a LEVEL II COBOL word, a literal, a PICTURE character-string, or a comment-entry. A character-string is delimited by separators.

index-names
library-names
mnemonic-names
paragraph-names
program-names
routine-names
section-names
text-names

All user-defined words, except segment-numbers and level-numbers, can belong to one and only one of these disjoint sets. Further, all user-defined words within a given disjoint set must be unique. (See **Uniqueness Of Reference** in this Section.)

With the exception of paragraph-name, section-name, level-number and segment-number, all user-defined words must contain at least one alphabetic character. Segment-numbers and level-numbers need not be unique; a given specification of a segment-number or level-number may be identical to any other segment-number or level-number and may even be identical to a paragraph-name or section-name.

Condition-Name: A condition-name is a name which is assigned to a specific value, set of values, or range of values, within a complete set of values that a data item may assume. The data item itself is called a conditional variable.

Condition-names may be defined in the Data Division or in the SPECIAL-NAMES paragraph within the Environment Division where a condition-name must be assigned to the ON STATUS or OFF STATUS, or both, of the run time switches.

A condition-name is used only in the RERUN clause or in conditions as an abbreviation for the relation condition; this relation condition posits that the associated conditional variable is equal to one of the set of values to which that condition-name is assigned.

Mnemonic-Name: A mnemonic-name assigns a user-defined word to an implementor-name. These associations are established in the SPECIAL-NAMES paragraph of the Environment Division. (See **SPECIAL-NAMES** in Chapter 3).

There are six types of reserved words:

1. Key words
2. Optional words
3. Connectives
4. Special registers
5. Figurative constants
6. Special-character words

Key Words: A key word is a word whose presence is required when the format in which the word appears is used in a source program. Within each format, such words are uppercase and underlined.

Key words are of three types:

1. Verbs such as ADD, READ, and ENTER.
2. Required words, which appear in statement and entry formats.
3. Words which have a specific functional meaning such as NEGATIVE, SECTION, etc.

Optional Words: Within each format, uppercase words that are not underlined are called optional words and may appear at the user's option. The presence or absence of an optional word does not alter the semantics of the COBOL program in which it appears.

Connectives: There are three types of connectives:

1. Qualifier connectives that are used to associate a data-name, a condition-name, and a text-name, or a paragraph-name with its qualifier: OF, IN.

2. Series connectives that link two or more consecutive operands: , (separator comma) or ; (separator semicolon).

3. Logical connectives that are used in the formation of conditions: AND, OR.

All other punctuation characters are part of the value of the nonnumeric literal rather than separators; all nonnumeric literals are category alphanumeric. (See **The PICTURE Clause** in chapter 3). In addition, hexadecimal binary values can be attributed to non numeric literals by expressing literals as: X"*nn*", where *n* is a hexadecimal character in the set 0-9 A-F; *nn* may be repeated up to 128 times, but the number of hexadecimal digits must be even.

Numeric Literals: A numeric literal is a character-string whose characters are selected from the digits '0' through '9', the plus sign, the minus sign, and/or the decimal point. The implementation allows for numeric literals of 1 through 18 digits in length. The rules for the formation of numeric literals are as follows:

1. A literal must contain at least one digit.

2. A literal must not contain more than one sign character. If a sign is used, it must appear as the leftmost character of the literal. If the literal is unsigned, the literal is positive.

3. A literal must not contain more than one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, the literal is an integer.

If a literal conforms to the rules for the formation of numeric literals, but is enclosed in quotation marks, it is a nonnumeric literal and it is treated as such by the compiler.

4. The value of a numeric literal is the algebraic quantity represented by the characters in the numeric literal. Every numeric literal is category numeric. (See **The PICTURE Clause** in Chapter 3).

The size of a numeric literal in standard data format characters is equal to the number of digits specified by the user.

When a figurative constant represents a string of one or more characters, the length of the string is determined by the compiler from context according to the following rules:

1. When a figurative constant is associated with another data item, as when the figurative constant is moved to or compared with another data item, the string of characters specified by the figurative constant is repeated character by character on the right until the size of the resultant string is equal to the size in characters of the associated data item. This is done prior to and independent of the application of any JUSTIFIED clause that may be associated with the data item.

2. When a figurative constant is not associated with another data item, as when the figurative constant appears in a DISPLAY, STRING, STOP or UNSTRING statement, the length of the string is one character.

DISPLAY SPACE in Format 2 of the DISPLAY statement is, of course, an exception.

A figurative constant may be used wherever a literal appears in a format, except that whenever the literal is restricted to having only numeric characters in it, the only figurative constant permitted is ZERO (ZEROS, ZEROES).

When the figurative constants HIGH-VALUE(S) or LOW-VALUE(S) are used in the source program, the actual character associated with each figurative constant depends upon the program collating sequence specified. (See **The OBJECT-COMPUTER Paragraph,** and **The SPECIAL-NAMES Paragraph** in Chapter 3).

Each reserved word which is used to reference a figurative constant value is a distinct character-string with the exception of the construction 'ALL literal' which is composed of two distinct character-strings.

PICTURE Character-Strings

A PICTURE character-string consists of certain combinations of characters in the COBOL character set used as symbols. See **The PICTURE Clause** in chapter 3 for the PICTURE character-string and for the rules that govern their use.

Level-Numbers

A system of level-numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, level-numbers for records start at 01. Less inclusive data items are assigned higher (not necessarily successive) level-numbers not greater in value than 49. A maximum of 49 levels in a record is allowed. There are special level-numbers, 66, 77 and 88 which are exceptions to this rule (see below). Separate entries are written in the source program for each level-number used.

A group includes all group and elementary items following it until a level-number less than or equal to the level-number of that group is encountered. All items which are immediately subordinate to a given group item must be described using identical level-numbers greater than the level-number used to describe that group item. Note that elementary items must not exceed 8191 bytes in length. Group items may defined as larger (for example for the purpose of being the parent items for large tables) but in such cases they must not be explicitly referenced.

Three types of entries exist for which there is no true concept of level. These are:

1. Entries that specify elementary items or groups introduced by a RENAMES clause

2. Entries that specify noncontiguous working storage and linkage data items

3. Entries that specify condition-names.

Entries describing items by means of RENAMES clauses for the purpose of regrouping data items have been assigned the special level-number 66.

Entries that specify noncontiguous data items, which are not subdivisions of other items, and are not themselves subdivided, have been assigned the special level-number 77.

Entries that specify condition-names, to be associated with particular values of a conditional variable, have been assigned the special level-number 88.

2. As numeric characters defined by USAGE IS DISPLAY (See **The USAGE Clause** in Chapter 3) one per byte in ASCII representation. If they are signed and the sign is specified as INCLUDED, bit 6 of the leading or trailing byte of the field is set for negative, depending on the field definition. If a SEPARATE sign is specified as a one byte ASCII + or -, a sign is added as the leading or trailing byte. If no SIGN clause is specified, bit 6 of the trailing digit is set to indicate negative by default.

3. As numeric characters defined by USAGE IS COMP or COMPUTATIONAL in pure binary form. If the field is signed the number is held in its twos-complement form. Storage is then dependent on the number of 9's in the PICTURE clause (see **The PICTURE Clause** in Chapter 3) and on whether the field is SIGNed or not (see **The SIGN Clause** in Chapter 3).

   Table 2-3 shows the storage requirements for each COMP(UTATIONAL) PICTURE clause.

Table 2-3. Numeric Data Storage for the COMP(UTATIONAL) PICTURE Clause.

Bytes Required	Number of Characters	
	Signed	Unsigned
1	1-2	1-2
2	3-4	3-4
3	5-6	5-7
4	7-9	8-9
5	10-11	10-12
6	12-14	13-14
7	15-16	15-16
8	17-18	17-18

4. As numeric characters defined by USAGE IS COMPUTATIONAL-3 or USAGE IS COMP-3 in packed internal decimal form. Storage is dependent on the number of 9's in the PICTURE clause. The decimal numbers are stored as signed strings of variable length of 1 through 18 digits. The sign of the packed decimal number is always stored in place of the least significant quartet of the low order byte. Each byte contains two decimal positions (four bits per digit) and the digits (0 - 9) are encoded as BCD numbers (0000 - 1001). Numbers are represented in the field as right-justified values with a + or - sign as shown in the example below. The maximum number of digits permitted in arithmetic operands is 18.

ALGEBRAIC SIGNS

Algebraic signs fall into two categories: operational signs, which are associated with signed numeric data items and signed numeric literals to indicate their algebraic properties; and editing signs, which appear on edited reports to identify the sign of the item.

The SIGN clause permits the programmer to state explicitly, the location of the operational sign. The clause is optional; if it is not used operational signs will be represented as defined by setting bit 6 of the trailing digit for ASCII numbers. (see above).

Editing signs are inserted into a data item through the use of the sign control symbols of the PICTURE clause.

STANDARD ALIGNMENT RULES

The standard rules for positioning data within an elementary item depend on the category of the receiving item. These rules are:

1. If the receiving data item is described as numeric:

    a. The data is aligned by decimal point and is moved to the receiving character positions with zero fill or truncation on either end as required.

    b. When an assumed decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately following its rightmost character and is aligned as in paragraph a. above.

2. If the receiving data item is a numeric edited data item, the data moved to the edited item is aligned by decimal point with zero fill or truncation at either end as required within the receiving character positions of the data item, except where editing requirements cause replacement of the leading zeros.

3. If the receiving data item is alphanumeric (other than a numeric edited data item), alphanumeric edited or alphabetic, the sending data is moved to the receiving character positions and aligned at the leftmost character position in the data item with space fill or truncation to the right, as required.

If the JUSTIFIED clause is specified for the receiving item, these standard rules are modified as described in **The JUSTIFIED Clause** in Chapter 3.

The general formats for qualification are:

Format 1

```
{data-name-1     }      [{OF} data-name-2 ]  ...
{condition-name  }       {IN}
```

Format 2

```
paragraph-name      [{OF} section-name ]
                     {IN}
```

Format 3

```
text-name           [{OF} library-name ]
                     {IN}
```

The rules for qualification are as follows:

1. Each qualifier must be of a successively higher level and within the same hierarchy as the name it qualifies.

2. The same name must not appear at two levels in a hierarchy.

3. If a data-name or a condition-name is assigned to more than one data item in a source program, the data-name or condition-name must be qualified each time it is referred to in the Procedure, Environment, and Data Divisions (except in the REDEFINES clause where qualification is unnecessary and must not be used.)

4. A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to from within the same section.

5. A data-name cannot be subscripted when it is being used as a qualifier.

The format is:

```
{data-name     }    (subscript-1[, subscript-2[, subscript-3 ] ... ] )
{condition name}
```

Indexing

References can be made to individual elements within a table of like elements by specifying indexing for that reference. An index is assigned to that level of the table by using the INDEXED BY phrase in the definition of a table. A name given in the INDEXED BY phrase is known as an index-name and is used to refer to the assigned index. The value of an index corresponds to the occurrence number of an element in the associated table or any other table. An index-name must be initialized before it is used as a table reference. An index-name can be given an initial value by a SET statement.

Direct indexing is specified by using an index-name in the form of a subscript. Relative indexing is specified when the index-name is followed by the operator + or -, followed by an unsigned integer numeric literal all delimited by the balanced pair of separators left parenthesis and right parenthesis following the table element data-name. The occurrence number resulting from relative indexing is determined by incrementing (where the operator + is used) or decrementing (when the operator - is used), by the value of the literal, the occurrence number represented by the value of the index. When more than one index-name is required, they are written in the order of successively less inclusive dimensions of the data organization.

At the time of execution of a statement which refers to an indexed table element, the value contained in the index referenced by the index-name associated with the table element must neither correspond to a value less than one nor to a value greater than the highest permissible occurrence number of an element of the associated table. This restriction also applies to the value resultant from relative indexing. In practice there is a limit to the number of index-names that can be used with a data-name. This is dependent on the length of the index-names but is approximately nine. If exceeded, internal buffer overflow results in a compile-time error.

3. An index may be modified only by the SET, SEARCH, and PERFORM statements. Data items described by the USAGE IS INDEX clause permit storage of the values associated with index-names as data in a form specified by the implementor. Such data items are called index data items.

4. literal-1, literal-3, literal-5, in the above format must be positive numeric integers. literal-2, literal-4, literal-6 must be unsigned numeric integers.

Condition-Name

Each condition-name must be unique, or be made unique through qualification and/or indexing, or subscripting. If qualification is used to make a condition-name unique, the associated conditional variable may be used as the first qualifier. If qualification is used, the hierarchy of names associated with the conditional variable or the conditional variable itself must be used to make the condition-name unique.

If references to a conditional variable require indexing or subscripting, then references to any of its condition-names also require the same combination of indexing or subscripting.

The format and restrictions on the combined use of qualification, subscripting, and indexing of condition-names is exactly that of identifier except that data-name-1 is replaced by condition-name-1.

Explicit and Implicit Specifications

There are three types of explicit and implicit specifications that occur in COBOL source programs:

1. Explicit and implicit Procedure Division references

2. Explicit and implicit transfers of control

3. Explicit and implicit attributes.

In addition to the implicit transfer of control between consecutive statements, implicit transfer of control also occurs when the normal flow is altered without the execution of a procedure branching statement. COBOL provides the following types of implicit control flow alterations which override the statement-to-statement transfers of control:

1. If a paragraph is being executed under control of another COBOL statement (for example, PERFORM, USE, SORT and MERGE) and the paragraph is the last paragraph in the range of the controlling statement, then an implied transfer of control occurs from the last statement in the paragraph to the control mechanism of the last executed controlling statement. Further, if a paragraph is being executed under the control of a PERFORM statement which causes iterative execution and that paragraph is the first paragraph in the range of that PERFORM statement, an implicit transfer of control occurs between the control mechanism associated with that PERFORM statement and the first statement in that paragraph for each iterative execution of the paragraph.

2. When a SORT or MERGE statement is executed, an implicit transfer of control occurs to any associated input or output procedures.

3. When any COBOL statement is executed which results in the execution of a declarative section, an implicit transfer of control to the declarative section occurs. Note that another implicit transfer of control occurs after execution of the declarative section, as described in (1) above.

4. In any file operation (including OPEN and CLOSE), if a file does not have a FILE STATUS data item declared for it and the file is not explicitly covered by a USE statement then it is covered by an implicit USE statement. The implied USE procedure is equivalent to:

```
USE AFTER ERROR PROCEDURE ON file-name
IF status-key-1 = 9
        DISPLAY error-message UPON CONSOLE
        STOP RUN.
```

where error-message is defined in the **LEVEL II COBOL Operating Guide.**

PROGRAM STRUCTURE

A LEVEL II COBOL program consists of four divisions:

1. IDENTIFICATION DIVISION - An identification of the program.

2. ENVIRONMENT DIVISION - A description of the equipment to be used to compile and run the program.

3. DATA DIVISION - A description of the data to be processed.

4. PROCEDURE DIVISION - A set of procedures to specify the operations to be performed on the data.

Each division is divided into sections which are further divided into paragraphs, which in turn are made up of sentences.

Within these subdivisions of a COBOL program, further subdivisions exist as clauses and statements. A clause is an ordered set of COBOL elements that specify an attribute of an entry, and a statement is a combination of elements in the Procedure Division that include a COBOL verb and constitute a program instruction.

LEVEL II COBOL Optional Syntax

Some of the 'red-tape' statements required by a strict ANSI interpretation are optional under LEVEL II COBOL. In the syntax descriptions in this manual these statements are marked ⊰ ⊱

If the operator enters the FLAG directive at compile time all LEVEL II COBOL extensions are flagged on the compiler listing together with ANSI COBOL requirements depending on their level as specified by the Federal Software Testing Center under the direction of the General Services Administration (GSA). See the description of the compiler FLAG directive in the LEVEL II COBOL Operating Guide.

ENVIRONMENT DIVISION

General Description

The Environment Division specifies a standard method of expressing those aspects of a data processing problem that are dependent upon the physical characteristics of a specific computer. This division allows specification of the configuration of the compiling computer and the object computer. In addition, information relating to input-output control, special hardware characteristics and control techniques can be given.

The Environment Division must be included in every COBOL source program.

Organization

Two sections make up the Environment Division: the Configuration Section and the Input-Output Section.

The Configuration Section deals with the characteristics of the source computer and the object computer. This section is divided into three paragraphs: the SOURCE-COMPUTER paragraph, which describes the computer configuration on which the source program is compiled; the OBJECT-COMPUTER paragraph, which describes the computer configuration on which the object program produced by the compiler is to be run; and the SPECIAL-NAMES paragraph, which relates the implementation-names used by the compiler to the mnemonic-names used in the source program.

The Input-Output Section deals with the information needed to control transmission and handling of data between external media and the object program. This section is divided into two paragraphs: the FILE-CONTROL paragraph which names and associates the files with external media; and the I-O-CONTROL paragraph which defines special control techniques to be used in the object program.

Structure

The following is the general format of the sections and paragraphs in the Environment Division, and defines the order of presentation in the source program.

The FILE SECTION defines the structure of data files. Each file is defined by a file description entry and one or more record descriptions. Record descriptions are written immediately following the file description entry. The WORKING-STORAGE SECTION describes records and noncontiguous data items which are not part of external data files but are developed and processed internally. It also describes data items whose values are assigned in the source program and do not change during the execution of the object program. The LINKAGE SECTION appears in the called program and describes data items that are to be referred to by the calling program and the called program. Its structure is the same as the WORKING-STORAGE SECTION. The communication section describes the data item in the source program that will serve as the interface between the MCS (Message Control System) and the program.

GENERAL FORMAT

The following gives the general format of the sections in the Data Division, and defines the order of their presentation in the source program.

```
{DATA DIVISION.  }

[ FILE SECTION.
[file-description-entry    [record-description-entry]    ...        ] ...  ]
[sort-merge-file-description-entry [record-description-entry] ...]

[ WORKING-STORAGE SECTION.
[77-level-description-entry ] ...]
[record-description-entry   ]

[ LINKAGE SECTION.
[77-level-description-entry] ...]
[record-description-entry  ]

[ COMMUNICATION SECTION.                                          ]
[communication-description-entry [record-description-entry] ...] ...]
```

A paragraph consists of a paragraph-name followed by a period and a space and by zero, one, or more successive sentences. A paragraph ends immediately before the next paragraph-name or section-name or at the end of the Procedure Division or, in the declaratives portion of the Procedure Division, at the key words END DECLARATIVES.

A sentence consists of one or more statements and is terminated by a period followed by a space.

A statement is a syntactically valid combination of words and symbols beginning with a COBOL verb.

The term *identifier* is defined as the word or words necessary to make unique reference to a data item.

EXECUTION

Execution begins with the first statement of the Procedure Division, excluding declaratives. Statements are then executed in the order in which they are presented for compilation, except where the rules indicate some other order.

GENERAL FORMAT

Procedure Division Header

The Procedure Division is identified by and must begin with the following header:

```
PROCEDURE DIVISION  [ USING data-name-1  [, data-name-2]  ...  ].
```

CONDITIONAL STATEMENT

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

A conditional statement is one of the following:

- An IF, SEARCH or RETURN statement.
- A READ statement that specifies the AT END or INVALID KEY phrase.
- A WRITE statement that specifies the INVALID KEY or END-OF-PAGE phrase.
- A START, REWRITE or DELETE statement that specifies the INVALID KEY phrase.
- An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) that specifies the SIZE ERROR phrase.
- A RECEIVE statement that specifies a NO DATA phrase.
- A STRING, UNSTRING or CALL statement that specifies the ON OVERFLOW phrase.

CONDITIONAL SENTENCE

A conditional sentence is a conditional statement, optionally preceded by an imperative statement, terminated by a period followed by a space.

COMPILER DIRECTING STATEMENT

A compiler directing statement consists of a compiler directing verb and its operands. The compiler directing verbs are COPY, ENTER and USE (see **The COPY Statement** in Chapter 10, **The ENTER Statement** in Chapter 3, and **The USE Statement** in Chapters 5, 6 and 7). A compiler directing statement causes the compiler to take a specified action during compilation.

1 - Without the optional SIZE ERROR phrase.
2 - Without the optional INVALID KEY phrase.
3 - Without the optional ON OVERFLOW phrase.
4 - Without the optional NO DATA phrase.
5 - Without the optional AT END phrase or INVALID KEY phrase.
6 - Without the optional INVALID KEY phrase or END-OF-PAGE phrase.

When 'imperative-statement' appears in the general format of statements, 'imperative-statement' refers to that sequence of consecutive imperative statements that must be ended by a period or an ELSE phrase associated with a previous IF statement.

IMPERATIVE SENTENCE

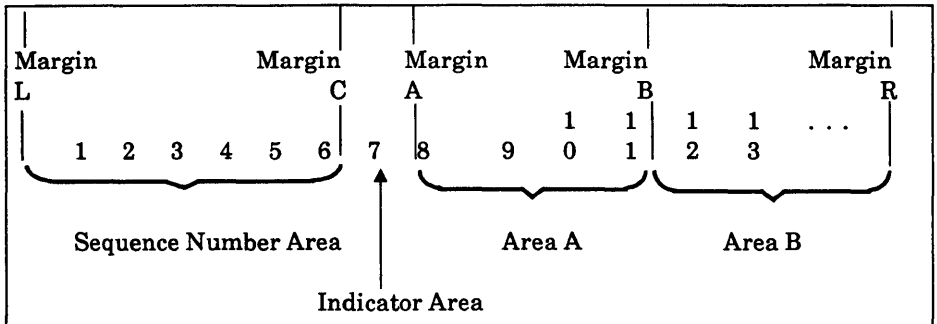
An imperative sentence is an imperative statement terminated by a period followed by a space.

Category	Verbs
Input-Output	ACCEPT (identifier) CLOSE DELETE DISABLE DISPLAY ENABLE OPEN READ RECEIVE REWRITE SEND START STOP (literal) WRITE
Inter-Program Communicating	CALL CANCEL
Ordering	MERGE RELEASE RETURN SORT
Procedure Branching	ALTER CALL EXIT GO TO PERFORM
Table Handling	SEARCH SET

IF is a verb in the COBOL sense; it is recognized that it is not a verb in English.

Reference Format Representation

The reference format for a line is represented as in Figure 2-1.



Margin L is immediately to the left of the leftmost character position of a line.

Margin C is between the 6th and 7th character positions of a line.

Margin A is between the 7th and 8th character positions of a line.

Margin B is between the 11th and 12th character positions of a line.

Margin R is immediately to the right of the rightmost character position of a line.

Figure 2-1. Reference Format for a COBOL Source Line.

The sequence number area occupies six character positions (1-6), and is between Margin L and Margin C.

The indicator area is the 7th character position of a line.

Area A occupies character positions 8, 9, 10 and 11, and is between margin A and margin B.

Area B occupies character positions 12 through 72 inclusive; it begins immediately to the right of Margin B and terminates immediately to the left of Margin R.

PSEUDO-TEXT

The character-strings and separators comprising pseudo-text may start in either area A or area B. If, however, there is a hyphen in the indicator area of a line which follows the opening pseudo-text delimiter, area A of the line must be blank; and the normal rules for continuation of lines apply to the formation of text words. (See Chapter 10, **Library**.)

Division, Section, Paragraph Formats

DIVISION HEADER

The division header must start in area A. (See Figure 2-1).

SECTION HEADER

The section header must start in area A. (See Figure 2-1).

A section consists of paragraphs in the Environment and Procedure Divisions and Data Division entries in the Data Division.

PARAGRAPH HEADER, PARAGRAPH-NAME AND PARAGRAPH

A paragraph consists of a paragraph-name followed by a period and a space and by zero, one or more sentences, or a paragraph header followed by one or more entries. Comment entries may be included within a paragraph. The paragraph header or paragraph-name starts in area A of any line following the first line of a division or a section.

The first sentence or entry in a paragraph begins either on the same line as the paragraph header or paragraph-name or in area B of the next nonblank line that is not a comment line. Successive sentences or entries either begin in area B of the same line as the preceding sentence or entry or in area B of the next nonblank line that is not a comment line.

Successive data description entries may have the same format as the first or may be indented according to level-number. The entries in the output listing need be indented only if the input is indented. Indentation does not affect the magnitude of a level-number.

When level-numbers are to be indented, each new level-number may begin any number of spaces to the right of margin A. The extent of indentation to the right is determined only by the width of the physical medium.

Declaratives

The key word DECLARATIVES and the key words END DECLARATIVES that precede and follow, respectively, the declaratives portion of the Procedure Division must each appear on a line by themselves. Each must begin in area A and be followed by a period and a space (see Figure 2-1).

Comment Lines

A comment line is any line with an asterisk in the continuation indicator area of the line. A comment line can appear as any line in a source program after the Identification Division header. Any combination of characters from the computer's character set may be included in area A and area B of that line (see Figure 2-1). The asterisk and the characters in area A and area B will be produced on the listing but serve as documentation only. A special form of comment line represented by a stroke in the indicator area of the line causes page ejection prior to printing the comment.

Successive comment lines are allowed. Continuation of comment lines is permitted, except that each continuation line must contain an '*' in the indicator area.

RESERVED WORDS

A full list of reserved words is given in the **LEVEL II COBOL Pocket Guide.**

CHAPTER 3

THE NUCLEUS

FUNCTION OF THE NUCLEUS

The Nucleus provides a basic language capability for the internal processing of data within the basic structure of the four divisions of a program.

OVERALL LANGUAGE

Name Characteristics

LEVEL II COBOL data-names need not begin with an alphabetic character; the alphabetic characters may be positioned anywhere within the data-name. Qualification is permitted and all data-names, condition-names, paragraph- names, and text-names need not be unique.

Figurative Constants

All the following figurative constants may be used: ZERO, ZEROS, ZEROES, SPACE, SPACES, HIGH-VALUE, HIGH-VALUES, LOW-VALUE, LOW-VALUES, QUOTE, QUOTES, and ALL literal.

Reference Format

A word or numeric literal can be broken in such a way that part of it appears on a continuation line.

SYNTAX RULES

1. The Identification Division must begin with the reserved words IDENTIFICATION DIVISION followed by a period and a space.

2. The comment-entry may be any combination of the characters from the computer's character set and may be written in Area B on one or more lines. The continuation of the comment-entry by the use of the hyphen in the indicator area is not permitted.

The PROGRAM-ID Paragraph

FUNCTION

The PROGRAM-ID paragraph gives the name by which a program is identified.

GENERAL FORMAT

```
PROGRAM-ID. program-name.
```

SYNTAX RULES

1. The program-name must conform to the rules for formation of a user-defined word.

GENERAL RULES

1. The PROGRAM-ID paragraph must contain the name of the program and must be present in every program in ANSI standard COBOL. In LEVEL II COBOL, however, this paragraph is optional.

2. The program-name identifies the source program and all listings pertaining to a particular program.

ENVIRONMENT DIVISION IN THE NUCLEUS

Configuration Section

THE SOURCE-COMPUTER PARAGRAPH

Function

The SOURCE-COMPUTER paragraph identifies the computer upon which the program is to be compiled.

General Format

```
SOURCE-COMPUTER.    computer-name.
```

Syntax Rule

computer-name must be one COBOL word defined by the user.

General Rules

The computer-name provides a means for identifying equipment configuration, in which case the computer-name and its implied configuration are specified by the user. The SOURCE-COMPUTER paragraph is used for documentary purposes only.

b. Explicitly specified in condition-name conditions; see **Condition Name Condition** (Conditional Variable).

3. If the PROGRAM COLLATING SEQUENCE clause is not specified, the native collating sequence is used. The **LEVEL II COBOL Pocket Guide** lists the full ASCII collating sequence (native) and those characters used in COBOL.

4. If the PROGRAM COLLATING SEQUENCE clause is specified, the program collating sequence is the collating sequence associated with the alphabet-name specified in that clause.

5. The PROGRAM COLLATING SEQUENCE clause is also applied to any nonnumeric merge or sort keys unless the COLLATING SEQUENCE phrase of the respective SORT or MERGE statement is specified.

6. The PROGRAM COLLATING SEQUENCE clause applies only to the program in which it is specified.

THE SPECIAL-NAMES PARAGRAPH

Function

The SPECIAL-NAMES paragraph provides a means of relating implementor-names to user-specified mnemonic-names and of relating alphabet-names to character sets and/or collating sequences.

Syntax Rules

1. mnemonic-names can be any COBOL user-defined word and at least one constituent character must be alphabetic.

2. The literals specified in the literal phrase of the alphabet-name clause:

   a. If numeric, must be unsigned integers and must have a value within the range of one (1) through the maximum number of characters in the native character set.

   b. If nonnumeric and associated with a THROUGH or ALSO phrase, must each be one character in length.

3. If the literal phrase of the alphabet-name clause is specified a given character must not be specified more than once in an alphabet-name clause.

4. The words THRU and THROUGH are equivalent.

5. data-name-2 must be defined in the Data Division as an alphanumeric data item, three characters in length.

6. data-name-2 may be a group item.

General Rules

1. function-name-1 specifies system devices or functions used by the compiler. The programmer can associate any user-defined COBOL word with a function-name. mnemonic-name-1, -2, etc can be used in the ACCEPT, DISPLAY or WRITE statements.

2. The SWITCH clause must have at least one condition-name associated with it. The switch is set at run-time by the operator and the setting may be determined in the program by testing the condition-names. The setting of the switches cannot be changed during execution.

If this clause is not present, only the currency sign is used in the PICTURE clause.

7. The clause DECIMAL-POINT IS COMMA means that the function of comma and period are exchanged in the character-string of the PICTURE clause and in numeric literals.

8. The clause CONSOLE IS changes the defaults in the ACCEPT and DISPLAY statements to the LEVEL II COBOL interactive extension that enables data to be accepted or displayed at any specified point on the screen. See **The ACCEPT Statement** and **The DISPLAY Statement** in this chapter.

9. The clause CURSOR IS data-name specifies the data-name to contain the CRT cursor address as used by the ACCEPT statement. If CURSOR IS is not specified the default cursor position on executing an ACCEPT statement is the 'Home' position at top left of the CRT screen. The CURSOR IS data-name clause enables a program to retain the position at the end of execution of the last ACCEPT statement or to specify the initial position at the start of any ACCEPT statement. This is a useful facility when programming menu-type operator prompts. The operator need then only move the cursor to the selected option prompt and press RETURN or just press RETURN for the default option.

   data-name contains the name either of a PIC 9999 field in which the most significant 99 represents a line count in the range one to the maximum number of lines on the user screen, and the least significant 99 represents a character position in the range one to the maximum positions allowed by the width of the user screen; or of a PIC 9(6) field in which the most significant 999 represents a line count in the range one to the maximum number of lines on the user screen, and the least significant 999 represents a character position in the range one to the maximum number of positions allowed by the user screen (this allows use of the right hand side of screens wider than 99 characters). If data-name is zero, the effect is as if the CURSOR IS clause was not used, that is, initial cursor position is top left of screen. (See also **The ACCEPT Statement** later in this chapter).

10. The SYSIN clause renames the system logical input unit (that is, the CRT keyboard).

11. The SYSOUT clause renames the system logical output unit (that is, the CRT screen).

Other data description clauses are optional and can be used to complete the description of the item if necessary.

WORKING-STORAGE RECORDS

Data elements and constants in Working-Storage which bear a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of record descriptions. All clauses which are used in record descriptions in the File Section can be used in record descriptions in the Working-Storage Section.

INITIAL VALUES

The initial value of any item in the Working-Storage Section except an index data item is specified by using the VALUE clause with the data item. The initial value of any index data item is unpredictable.

The Data Description-Complete Entry Skeleton

FUNCTION

A data description entry specifies the characteristics of a particular item of data.

SYNTAX RULES

1. The level-number in Format 1 may be any number from 01-49 or 77.

2. The clauses may be written in any order with two exceptions: the data-name-1 or FILLER clause must immediately follow the level-number; the REDEFINES clause, when used, must immediately follow the data-name-1 clause.

3. The PICTURE clause must be specified for every elementary item except an index data item, in which case use of this clause is prohibited.

4. The words THRU and THROUGH are equivalent.

GENERAL RULES

1. The clauses SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO, must not be specified except for an elementary data item.

2. Format 3 is used for each condition-name. Each condition-name requires a separate entry with level-number 88. Format 3 contains the name of the condition and the value, values, or range of values associated with the condition-name. The condition-name entries for a particular conditional variable must follow the entry describing the item with which the condition-name is associated. A condition-name can be associated with any data description entry which contains a level-number except the following:

   a. Another condition-name.

   b. A level 66 item.

   c. A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED or USAGE (other than USAGE IS DISPLAY).

   d. An index data item (See **The USAGE IS INDEX Clause** in Chapter 4).

GENERAL FORMAT

```
{data-name}
{FILLER   }
```

SYNTAX RULE

In the File, Working-Storage, Communication and Linkage Sections, a data-name or the key word FILLER must be the first word following the level-number in each data description entry.

GENERAL RULE

The key word FILLER may be used to name an elementary item or group in a record. Under no circumstances can a FILLER item be referred to explicitly. However, the key word FILLER may be used as a conditional variable because such use does not require explicit reference to the FILLER item but to its value.

The JUSTIFIED Clause

FUNCTION

The JUSTIFIED clause specifies non-standard positioning of data within a receiving data item.

GENERAL FORMAT

```
{JUSTIFIED} RIGHT
{JUST     }
```

GENERAL FORMAT

```
level-number
```

SYNTAX RULES

1. A level-number is required as the first element in each data description entry.

2. Data description entries subordinate to an FD, CD, or SD entry must have level-numbers with the values 01-49, 66 or 88. (See **The File Description** in Chapter 5).

3. Data description entries in the Working-Storage Section and Linkage Section must have level-numbers with the values 01-49, 66, 77 or 88.

4. A level number may be a one or two digit number.

GENERAL RULES

1. The level-number 01 identifies the first entry in each record description.

2. Special level numbers have been assigned to certain entries where there is no real concept of level:

   a. The level-number 77 is assigned to identify noncontiguous working storage data items, noncontiguous linkage data items, and can be used only as described by Format 1 of the data description skeleton. (See **The Data Description - Complete Entry Skeleton** in this chapter).

   b. Level number 66 is assigned to identify RENAMES entries and can be used only as described in Format 2 of the data description skeleton earlier in this chapter.

   c. Level number 88 is assigned to entries which define condition-names associated with a conditional variable and can be used only as described in Format 3 of the data description skeleton earlier in this chapter.

3. Multiple level 01 entries subordinate to any given level indicator, represent implicit redefinitions of the same area.

GENERAL RULES

There are five categories of data that can be described with a PICTURE clause: alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited. General rules within these categories are given below:

Alphabetic Data Rules

1. Its PICTURE character-string can only contain the symbols 'A', 'B'; and

2. Its contents when represented in standard data format must be any combination of the twenty-six (26) letters of the Roman alphabet and the space from the COBOL character set. Its length is limited only by the data division having to be less than 64K bytes long.

Numeric Data Rules

1. Its PICTURE character-string can only contain the symbols '9', 'P', 'S', and 'V'. The number of digit positions that can be described by the PICTURE character-string must range from 1 to 18 inclusive.

2. If unsigned, the data in standard data format must be a combination of the Arabic numerals '0', '1', '2', '3', '4', '5', '6', '7', '8', and '9'; if signed, the item may also contain a '+', '−', or other representation of an operational sign. (see **The SIGN Clause** later in this chapter).

Alphanumeric Data Rules

1. Its PICTURE character-string is restricted to certain combinations of the symbols 'A', 'X', '9', and the item is treated as if the character-string contained all X's. A PICTURE character-string which contains all A's or all 9's does not define an alphanumeric item; and

2. Its contents when represented in standard data format can consist of any characters in the computer's character set. Its length is limited only by the data division having to be less than 64K bytes long.

Elementary Item Size

The size of an elementary item, where size means the number of character positions occupied by the elementary item in standard data format, is determined by the number of allowable symbols that represent character positions. An integer which is enclosed in parentheses following the symbols 'A', ',', 'X', '9', 'P', 'Z', '*', 'B', '/', '0', '+', '−', or the currency symbol indicates the number of consecutive occurrences of the symbol. Note that the following symbols may appear only once in a given PICTURE: 'S', 'V', '.', 'CR', and 'DB'.

Symbols Used

The functions of the symbols used to describe an elementary item are explained as follows:

A Each 'A' in the character-string represents a character position which can contain only a letter of the alphabet or a space.

B Each 'B' in the character-string represents a character position into which the space character will be inserted.

P Each 'P' indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character 'P' is not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items or numeric items. The scaling position character 'P' can appear only to the left or right as a continuous string of 'P's within a PICTURE description; since the scaling position character 'P' implies an assumed decimal point (to the left of 'P's if 'P's are leftmost PICTURE characters and to the right if 'P's are rightmost PICTURE characters), the assumed decimal point symbol 'V' is redundant as either the leftmost or rightmost character within such a PICTURE description. The character 'P' and the insertion character '.' (period) cannot both occur in the same PICTURE character-string. If, in any operation involving conversion of data from one form of internal representation to another, the data item being converted is described with the PICTURE character 'P', each digit position described by a 'P' is considered to contain the value zero, and the size of the data item is considered to include the digit positions so described.

. When the character '.' (period) appears in the character-string it is an editing symbol which represents the decimal point for alignment purposes and in addition, represents a character position into which the character '.' will be inserted. The character '.' is counted in the size of the item. For a given program the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange the rules for the period apply to the comma and the rules for the comma apply to the period wherever they appear in a PICTURE clause. The insertion character '.' must not be the last character in the PICTURE character-string.

+, -, CR, DB

These symbols are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in any one character-string and each character used in the symbol is counted in determining the size of the data item.

* Each '*' (asterisk) in the character-string represents a leading numeric character position into which an asterisk will be placed when the contents of that position is zero. Each '*' is counted in the size of the item.

cs The currency symbol in the character-string represents a character position into which a currency symbol is to be placed. The currency symbol in a character-string is represented by either the currency sign or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. The currency symbol is counted in the size of the item.

EDITING RULES

There are two general methods of performing editing in the PICTURE clause, either by insertion or by suppression and replacement. There are four types of insertion editing available. They are:

- Simple insertion
- Special insertion
- Fixed insertion
- Floating insertion

Special Insertion Editing

The '.' (period) is used as the insertion character. In addition to being an insertion character it also represents the decimal point for alignment purposes. The insertion character used for the actual decimal point is counted in the size of the item. The use of the assumed decimal point, represented by the symbol 'V' and the actual decimal point, represented by the insertion character, in the same PICTURE character-string is disallowed. The result of special insertion editing is the appearance of the insertion character in the item in the same position as shown in the character-string. *

Fixed Insertion Editing

The currency symbol and the editing sign control symbols, '+', '−', 'CR', 'DB', are the insertion characters. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character-string. When the symbols 'CR' or 'DB' are used they represent two character positions in determining the size of the item and they must represent the rightmost character positions that are counted in the size of the item. The symbol '+' or '−', when used, must be either the leftmost or rightmost character position to be counted in the size of the item. The currency symbol must be the leftmost character. *

Table 3-2. Editing Symbols in PICTURE Character-Strings

EDITING SYMBOL IN PICTURE CHARACTER-STRING	RESULT	
	DATA ITEM POSITIVE OR ZERO	DATA ITEM NEGATIVE
+	+	−
−	space	−
CR	2 spaces	CR
DB	2 spaces	DB

Zero Suppression Editing

The suppression of leading zeros in numeric character positions is indicated by the use of the alphabetic character 'Z' or the character '*' (asterisk) as suppression symbols in a PICTURE character-string. These symbols are mutually exclusive in a given PICTURE character-string. Each suppression symbol is counted in determining the size of the item. If 'Z' is used, the replacement character will be the space and if the asterisk is used, the replacement character will be '*'.

Zero suppression and replacement is indicated in a PICTURE character-string by using a string of one or more of the allowable symbols to represent leading numeric character positions which are to be replaced when the associated character position in the data contains a zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string.

In a PICTURE character-string, there are only two ways of representing zero suppression. One way is to represent any or all of the leading numeric character positions to the left of the decimal point by suppression symbols. The other way is to represent all of the numeric character positions in the PICTURE character-string by suppression symbols.

If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates at the first non-zero digit in the data represented by the suppression symbol string or at the decimal point, whichever is encountered first.

If all numeric character positions in the PICTURE character-string are represented by suppression symbols and the value of the data is not zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero and the suppression symbol is 'Z', the entire data item will be spaces. If the value is zero and the suppression symbol is '*', the data item will be all '*' except for the actual decimal point.

The symbols '+', '−', '*', 'Z', and the currency symbol, when used as floating replacement characters, are mutually exclusive within a given character-string.

Table 3-3. PICTURE Character Precedence Chart.

First Symbol / Second Symbol		Non-Floating Insertion Symbols									Floating Insertion Symbols						Other Symbols					
		B	0	/	,	.	{±}	{±}	{CR DB}	cs	{Z *}	{Z *}	{±}	{±}	cs	cs	9	A X	S	V	P	P
Non Floating Insertion Symbols	B	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	0	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	/	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	,	x	x	x	x	x	x			x	x	x	x	x	x	x	x			x		x
	.	x	x	x	x		x			x	x		x		x		x					
	{±}																					
	{±}	x	x	x	x	x				x	x	x			x	x	x			x	x	x
	{CR DB}	x	x	x	x	x				x	x	x			x	x	x			x	x	x
	cs						x															
Floating Insertion Symbols	{Z *}	x	x	x	x		x			x	x											
	{Z *}	x	x	x	x	x	x			x	x	x								x		x
	{±}	x	x	x	x					x			x									
	{±}	x	x	x	x	x				x			x	x						x		x
	cs	x	x	x	x		x								x							
	cs	x	x	x	x	x	x								x	x				x		x
Other Symbols	9	x	x	x	x	x	x			x	x		x		x		x	x	x	x		x
	A X	x	x	x													x	x				
	S																					
	V	x	x	x	x		x			x	x		x		x		x		x		x	
	P	x	x	x	x		x			x	x		x		x		x		x		x	
	P						x			x									x	x		x

3. This clause must not be used in level 01 entries in the File Section. (See General Rule 2 of **The DATA RECORDS Clause** in Chapter 5.)

4. This clause must not be used in level 01 entries in the Communication Section.

5. Data-name-2 may be subordinate to an entry which contains a REDEFINES clause. The data description entry for data-name-2 cannot contain an OCCURS clause. However data-name-2 may be subordinate to an item whose data description entry contains an OCCURS clause. In this case, the reference to data-name-2 in the REDEFINES clause may not be subscripted or indexed. Neither the original definition nor the redefinition can include an item whose size is variable as defined in the OCCURS clause. (See **The OCCURS Clause** in Chapter 4).

6. No entry having a level-number numerically lower than the level-number of data-name-2 and data-name-1 may occur between the data description entries of data-name-2 and data-name-1.

GENERAL RULES

1. Redefinition starts at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered.

2. When the level-number of data-name-1 is other than 01, it must specify the same number of character positions that the data item referenced by data-name-2 contains in ANSI standard COBOL. It is important to observe that the REDEFINES clause specifies the redefinition of a storage area, not of the data items occupying the area.

3. Multiple redefinitions of the same character positions are permitted. The entries giving the new descriptions of the character positions must follow the entries defining the area being redefined, without intervening entries that define new character positions. Multiple redefinitions of the same character positions must all use the data-name of the entry that originally defined the area.

4. The entries giving the new description of the character positions must not contain any VALUE clauses except in condition-name entries.

3. data-name-1 cannot be used as a qualifier, and can only be qualified by the names of the associated level 01, FD, CD or SD entries. Neither data-name-2 nor data-name-3 may have an OCCURS clause in its data description entry nor be subordinate to an item that has an OCCURS clause in its data description entry. (See **The OCCURS Clause** in Chapter 4.)

4. The beginning of the area described by data-name-3 must not be to the left of the beginning of the area desribed by data-name-2. The end of the area described by data-name-3 must be the right of the end of the area described by data-name-2. data-name-3, therefore, cannot be subordinate to data-name-2.

5. data-name-2 and data-name-3 may be qualified.

6. The words THRU and THROUGH are equivalent.

7. None of the items within the range, including data-name-2 and data-name-3, if specified, can be an item whose size is variable as defined in **The OCCURS Clause** in Chapter 4.

GENERAL RULES

1. One or more RENAMES entries can be written for a logical record.

2. When data-name-3 is specified, data-name-1 is a group item which includes all elementary items starting with data-name-2 (if data-name-2 is an elementary item) or the first elementary item in data-name-2 (if data-name-2 is a group item), and concluding with data-name-3 (if data-name-3 is an elementary item) or the last elementary item in data-name-3 (if data-name-3 is a group item).

3. When data-name-3 is not specified, data-name-2 can be either a group or an elementary item; when data-name-2 is a group item, data-name-1 is treated as a group item, and when data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

GENERAL RULES

1. The optional SIGN clause, if present, specifies the position and the mode of representation of the operational sign for the numeric data description entry to which it applies, or for each numeric data description entry subordinate to the group to which it applies. The SIGN clause applies only to numeric data description entries whose PICTURE contains the character 'S'; the 'S' indicates the presence of, but neither the representation nor, necessarily, the position of the operational sign.

2. A numeric data description entry whose PICTURE contains the character 'S', but to which no optional SIGN clause applies, has an operational sign, but neither the representation nor, necessarily, the position of the operational sign is specified by the character 'S'. In this (default) case, general rules 3 through 5 do not apply to such signed numeric data items. The representation of the default operational sign is defined in Chapter 2 under the heading **Selection of Character Representation and Radix**.

3. If the optional SEPARATE CHARACTER phrase is not present, then:

   a. The operational sign will be presumed to be associated with the leading (or, respectively, trailing) digit position of the elementary numeric data item.

   b. The letter 'S' in a PICTURE character-string is not counted in determining the size of the item (in terms of standard data format characters).

4. If the optional SEPARATE CHARACTER phrase is present, then:

   a. The operational sign will be presumed to be the leading (or, respectively, trailing) character position of the elementary numeric data item; this character position is not a digit position.

   b. The letter 'S' in a PICTURE character-string is counted in determining the size of the item (in terms of standard data format characters).

   c. The operational signs for positive and negative are the standard data format characters '+' and '−', respectively.

5. Every numeric data description entry whose PICTURE contains the character 'S' is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

2. This clause specifies that the subject data item is to be aligned in the computer such that no other data item occupies any of the character positions between the leftmost and rightmost natural boundaries delimiting this data item. If the number of character positions required to store this data item is less than the number of character positions between those natural boundaries, the unused character positions (or portions thereof) must not be used for any other data item. Such unused character positions, however, are included in:

   a. The size of any group item(s) to which the elementary item belongs; and

   b. The character positions redefined when this data item is the object of a REDEFINES clause.

3. SYNCHRONIZED not followed by either RIGHT or LEFT specifies that the elementary item is to be positioned between natural boundaries in such a way as to effect efficient utilization of the elementary data item.

4. SYNCHRONIZED LEFT specifies that the elementary item is to be positioned such that it will begin at the left character position of the natural boundary in which the elementary item is placed.

5. SYNCHRONIZED RIGHT specifies that the elementary item is to be positioned such that it will terminate on the right character position of the natural boundary in which the elementary item is placed.

6. Whenever a SYNCHRONIZED item is referenced in the source program, the original size of the item, as shown in the PICTURE clause, is used in determining any action that depends on size, such as justification, truncation or overflow.

7. If the data description of an item contains the SYNCHRONIZED clause and an operational sign, the sign of the item appears in the normal operational sign position, regardless of whether the item is SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT.

8. When the SYNCHRONIZED clause is specified in a data description entry of a data item that also contains an OCCURS clause, or in a data description entry of a data item subordinate to a data description entry that contains an OCCURS clause, then:

   a. Each occurrence of the data item is SYNCHRONIZED.

2. This clause specifies the manner in which a data item is represented in the storage of a computer. It does not affect the use of the data item, although the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referred to. The USAGE clause may affect the radix or type of character representation of the item.

3. A COMPUTATIONAL or COMPUTATIONAL-3 item is capable of representing a value to be used in computations and must be numeric. If a group item is described as COMPUTATIONAL(-3), the elementary items in the group are COMPUTATIONAL(-3). The group item itself is not COMPUTATIONAL(-3) and cannot be used in computations.

4. The USAGE IS DISPLAY clause indicates that the format of the data is a standard data format.

5. If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, the usage is implicitly DISPLAY.

6. Space requirements for the various USAGE storage options are given under **Selection of Character Representation and Radix** in Chapter 2.

The VALUE Clause

FUNCTION

The VALUE clause defines the value of constants, the initial value of working storage items, and the values associated with a condition name.

GENERAL FORMAT

Format 1:

```
VALUE IS literal
```

b. If the category of the item is alphabetic, alphanumeric, alphanumeric edited or numeric edited, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric. (See **Standard Alignment Rules** in Chapter 2.) Editing characters in the PICTURE clause are included in determining the size of the data item (see **The PICTURE Clause** earlier in this chapter) but have no effect on initialization of the data item. Therefore, the VALUE for an edited item is presented in an edited form.

c. Initialization takes place independent of any BLANK WHEN ZERO or JUSTIFIED clause that may be specified.

2. A figurative constant may be substituted in both Format 1 and Format 2 wherever a literal is specified.

CONDITION-NAME RULES

1. In a condition-name entry, the VALUE clause is required. The VALUE clause and the condition-name itself are the only two clauses permitted in the entry. The characteristics of a condition-name are implicitly those of its conditional variable.

2. Format 2 can be used only in connection with condition-names. Wherever the THRU phrase is used, literal-1 must be less than literal-2, literal-3 less than literal-4, etc.

DATA DESCRIPTION ENTRIES OTHER THAN CONDITION-NAMES

Rules governing the use of the VALUE clause differ with the respective sections of the Data Division:

1. In the File Section, the VALUE clause may be used only in condition-name entries.

PROCEDURE DIVISION IN THE NUCLEUS

Arithmetic Expressions

DEFINITION OF AN ARITHMETIC EXPRESSION

An arithmetic expression can be an identifier of a numeric elementary item, a numeric literal, such identifers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Any arithmetic expression may be preceded by a unary operator. The permissible combinations of variables, numeric literals, arithmetic operator and parentheses are given in Table 3-4, Combination of Symbols in Arithmetic Expressions.

Those identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic may be performed.

ARITHMETIC OPERATORS

There are five binary arithmetic operators and two unary arithmetic operators that may be used in arithmetic expressions. They are represented by specific characters that may be preceded by a space and followed by a space.

3. The ways in which operators, variables, and parentheses may be combined in an arithmetic expression are summarized in Table 3-4, where:

   a. The letter 'P' indicates a permissible pair of symbols.

   b. The character '-' indicates an invalid pair.

   c. 'Variable' indicates an identifier or literal.

Table 3-4. Combination of Symbols in Arithmetic Expressions

FIRST SYMBOL	SECOND SYMBOL				
	Variable	*/** - +	Unary + or -	(	)
Variable	-	P	-	-	P
*/** + -	P	-	P	P	-
Unary + or -	P	-	P	P	-
(	P	-	P	P	-
)	-	P	-	-	P

4. An arithmetic expression may only begin with the symbol '(', '+', '-', or a variable and may only end with a ')' or a variable. There must be a one-to-one correspondence between left and right parentheses of an arithmetic expression such that each left parenthesis is to the left of its corresponding right parenthesis.

5. Arithmetic expressions allow the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items. See, for example, syntax rule 3 of **The ADD Statement** in this chapter.

The first operand (identifier-1, literal-1 or arithmetic-expression-1) is called the subject of the condition; the second operand (identifier-2 or literal-2 or arithmetic-expression-2) is called the object of the condition. The relation condition must contain at least one reference to a variable.

The relational operator specifies the type of comparison to be made in a relation condition. A space must precede and follow each reserved word comprising the relational operator. When used, 'NOT' and the next key word or relation character are one relational operator that defines the comparison to be executed for truth value; for example, 'NOT EQUAL' is a truth test for an 'unequal'.

Comparison: 'NOT GREATER' is a truth test for an 'equal' or 'less' comparison. The meaning of the relational operators is as shown in Table 3-5.

Table 3-5. Relational Operators.

Meaning	Relational Operator
Greater than or not greater than	IS [NOT]GREATER THAN IS [NOT] >
Less than or not less than	IS [NOT]LESS THAN IS [NOT] <
Equal to or not equal to	IS [NOT]EQUAL TO IS [NOT] =
The required relational characters '>', '<', and '=' are not underlined to avoid confustion with other symbols such as '≥' (greater than or equal to).	

Comparison of Numeric Operands: For operands whose class is numeric a comparison is made with respect to the algebraic value of the operands. The length of the literal or arithmetic expression operands in terms of number of digits represented, is not significant. Zero is considered a unique value regardless of the sign.

The first encountered pair of unequal characters is compared to determine their relative position in the collating sequence. The operand that contains the character that is positioned higher in the collating sequence is considered to be the greater operand.

2. Operands of unequal size - if the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by sufficient spaces to make the operands of equal size.

CLASS CONDITIONS

The class condition determines whether the operand is numeric, that is, consists entirely of the characters '0', '1', '2', '3', ..., '9', with or without the operational sign; or alphabetic, that is, consists entirely of the characters 'A', 'B', 'C', ..., 'Z', space. The general format for the class condition is as follows:

```
identifier IS [NOT]  { NUMERIC    }
                     { ALPHABETIC }
```

The usage of the operand being tested must be described as display. When used, 'NOT' and the next key word specify one class condition that defines the class test to be executed for truth value; for example, 'NOT NUMERIC' is a truth test for determining that an operand is nonnumeric.

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic or as a group item composed of elementary items whose data description indicates the presence of operational sign(s). If the data description of the item being tested does not indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and an operational sign is not present. If the data description of the item does indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and a valid operational sign is present. Valid operational signs for data items described with the SIGN IS SEPARATE clause are the standard data format characters, '+' and '−'.

The ALPHABETIC test cannot be used with an item whose data description describes the item as numeric. The item being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters 'A' through 'Z' and the space.

Sign Condition

The sign condition determines whether or not the algebraic value of an arithmetic expression is less than, greater than or equal to zero. The general format for a sign condition is as follows:

```
arithmetic-expression IS [NOT] { POSITIVE }
                               { NEGATIVE }
                               { ZERO     }
```

When used, 'NOT' and the next key word specify one sign condition that defines that algebraic test to be executed for truth value; for example, 'NOT ZERO' is a truth test for a nonzero (positive or negative) value. An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero. The arithmetic expression must contain at least one reference to a variable.

COMPLEX CONDITIONS

A complex condition is formed by combining simple conditions, combined conditions and/or complex conditions with logical connectors (logical operators 'AND' and 'OR') or negating these conditions with logical negation (the logical operator 'NOT'). The truth value of a complex condition, whether parenthesized or not, is that truth value which results from the interaction of all the stated logical operators on the individual truth values of simple conditions, or the intermediate truth values of conditions logically connected or logically negated.

where 'condition' may be:

a. A simple condition, or

b. A negated simple condition, or

c. A combined condition, or

d. A negated combined condition; that is, the 'NOT' logical operator followed by a combined condition enclosed within parentheses, or

e. Combinations of the above, specified according to the rules summarized in Table 3-6, Combinations of Conditions, Logical Operators, and Parentheses.

*

Although parentheses need never be used when either 'AND' or 'OR' (but not both) is used exclusively in a combined condition, parentheses may be used to effect a final truth value when a mixture of 'AND', 'OR' and 'NOT' is used. (See Table 3-6, Combinations of Conditions, Logical Operators, and Parentheses below, and Condition Evaluation Rules earlier in this chapter.)

Table 3-6 indicates the ways in which conditions and logical operators may be * combined and parenthesized. There must be a one-to-one correspondence between left and right parentheses such that each left parenthesis is matched by a corresponding right parenthesis. The table assumes a left to right sequence of elements.

Table 3-6. Combinations of Conditions, Logical Operators, and Parentheses

Element	Permitted location in conditional expression	Element can be preceded by only:	Element can be followed by only:
simple-condition	Any	OR, NOT, AND, (	OR, AND,)
OR, or AND	Not first or last	simple-condition,)	simple-condition, NOT, (
NOT	Not last	OR, AND, (	simple-condition, (
(	Not last	OR, NOT, AND, (	simple-condition, NOT, (
)	Not first	simple-condition,)	OR, AND,)

Some examples of abbreviated combined and negated combined relation conditions and expanded equivalents follow.

Abbreviated Combined Relation Condition	Expanded Equivalent
a > *b* AND NOT < *c* OR *d*	((*a* > *b*) AND (*a* NOT < *c*)) OR (*a* NOT < *d*)
a NOT EQUAL *b* OR *c*	(*a* NOT EQUAL *b*) OR (*a* NOT EQUAL *c*)
NOT *a* = *b* OR *c*	(NOT (*a* = *b*)) OR (*a* = *c*)
NOT (*a* GREATER *b* OR < *c*)	NOT ((*a* GREATER *b*) OR (*a* < *c*))
NOT (*a* NOT > *b* AND *c* AND NOT *d*)	NOT ((((*a* NOT > *b*) AND (*a* NOT > *c*)) AND(NOT (*a* NOT > *d*))))

Condition Evaluation Rules

Parentheses may be used to specify the order in which individual conditions of complex conditions are to be evaluated when it is necessary to depart from the implied evaluation precedence. Conditions within parentheses are evaluated first, and, within nested parentheses, evaluation proceeds from the least inclusive condition to the most inclusive condition. When parentheses are not used, or parenthesized conditions are at the same level of inclusiveness, the following hierarchical order of logical evaluation is implied until the final truth value is determined:

1. Values are established for arithmetic expressions. (See **Formation and Evaluation Rules** under **Arithmetic Expressions** in this chapter).

2. Truth values for simple conditions are established in the following order:

   relation (following the expansion of any abbreviated relation condition)
   class
   condition-name
   switch-status
   sign

3. Truth values for negated conditions are established.

THE SIZE ERROR PHRASE

If, after decimal point alignment, the absolute value of a result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. Division by zero always causes a size error condition. The size error condition applies only to the final results, except in MULTIPLY and DIVIDE statements; in which case the size error condition applies to the intermediate results as well. If the ROUNDED phrase is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action depends on whether or not the SIZE ERROR phrase is specified, as follows:

SIZE ERROR Phrase Not Specified

When a size error condition occurs, the value of those resultant-identifier(s) affected is undefined. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation.

SIZE ERROR Phrase Specified

When a size error condition occurs, then the values of resultant-identifier(s) affected by the size errors are not altered. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation. After completion of the execution of this operation, the imperative statement in the SIZE ERROR phrase is executed.

For the ADD statement with the CORRESPONDING phrase and the SUBTRACT statement with the CORRESPONDING phrase, if any of the individual operations produces a size error condition, the imperative statement in the SIZE ERROR phrase is not executed until all of the individual additions or subtractions are completed.

2. The maximum size of each operand is 18 decimal digits. The composite of operands, which is a hypothetical data item resulting from the superimposition of specified operands in a statement aligned on their decimal points (see **The ADD Statement, The DIVIDE Statement, The MULTIPLY Statement** and **The SUBTRACT Statement** later in this chapter) must not contain more than 18 decimal digits.

OVERLAPPING OPERANDS

When a sending and a receiving item in an arithmetic statement or an INSPECT, MOVE, SET, STRING or UNSTRING statement share a part of their storage areas, the result of the execution of such a statement is undefined.

MULTIPLE RESULTS IN ARITHMETIC STATEMENTS

The ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements may have multiple results. Such statements behave as though they had been written in the following way:

1. A statement which performs all arithmetic necessary to arrive at the result to be stored in the receiving items, and stores that result in a temporary storage location.

2. A sequence of statements transferring or combining the value of this temporary location with a single result. These statements are considered to be written in the same left-to-right sequence in which the multiple results are listed.

   The result of the statement

   ADD *a, b, c* TO *c, d (c), e*

The ACCEPT Statement

FUNCTION

The ACCEPT statement causes data keyed at the CRT console to be made available to the program in a specified data item.

GENERAL FORMATS

Format 1

```
ACCEPT identifier  [ FROM { mnemonic-name } ]
                          { CONSOLE       }
```

Format 2

```
ACCEPT data-name-1  { [ AT { data-name-2 } ] FROM CRT }
                           { literal-1   }
                    {   AT { data-name-2 }           }
                           { literal-1   }
```

Format 3

```
ACCEPT identifier FROM { DATE }
                       { DAY  }
                       { TIME }
```

SYNTAX RULES

The mnemonic-name in Format 1 must also be specified in the SPECIAL-NAMES paragraph of the Environment Division and must be associated with the console.

b. If the size of the transferred data record exceeds the size of the receiving data item (or the portion of the receiving data item not yet occupied by transferred data) only the leftmost characters of the input data are stored in the receiving data item (or the portion remaining). The remaining characters of the input data which do not fit into the receiving data item are ignored.

Format 2

6. The ACCEPT statement causes the transfer of data from the CRT to data-name-1. The contents of data-name-1 are replaced by this data.

7. data-name-1 is taken as a definition of the screen area in which elementary data items correspond to areas on the screen into which the operator can key information. FILLER fields correspond to areas on the screen which are inaccessible to the operator. data-name-1 must not be subscripted.

8. Elementary data items within data-name-1 may be alphanumeric, numeric usage display, or edited. The treatment and behaviour of edited fields depends on the configuration of the Accept-Display Module (ADIS). Refer to the details of the configuration utility in your **LEVEL II COBOL Operating Guide** or **Getting Started with LEVEL II COBOL** manual.

9. AT data-name-2 or literal-1 defines the position on the screen of the leftmost character of the data. Either form must refer to a PIC 9999 field or to a PIC 9(6) field. The most significant half is taken as a line count in the range one to the maximum lines on the user screen. The least significant half is taken as a character position in the range one to the maximum positions allowed by the screen width of the user CRT. Note that the size of this data-item is independent of the size of the CURSOR IS data-item.

10. data-name-1 may refer to a record, group or elementary item, but it may not be subscripted. REDEFINES may be used within data-name-1, in which case the first description of the data is used and subsequent descriptions are ignored. OCCURS and nested OCCURS may also be used with the effect that the repeated data-item is expanded into the full number of times it occurs and one definition is thus automatically repeated for many fields.

16. Before keying CR or equivalent, the operator can reposition the cursor to overwrite data already keyed or to skip character positions by use of the character positioning keys.

17. If the CRT STATUS IS data-name clause has been used in the SPECIAL-NAMES paragraph of the Environment Division, the data-item identified by the clause will be updated after every ACCEPT statement. The contents of this data item and their meanings are:

    1st Character
    (ASCII)

    0 Normal termination of the ACCEPT - either by the RETURN key or equivalent, or, if the Autoskip feature is enabled (see the **LEVEL II COBOL Operating Guide** or **Getting Started with LEVEL II COBOL** manual) by tabbing out of the last field. Character 2 and 3 are undefined.

    1 A user-defined function key was used to terminate the ACCEPT. Character 2 contains a COMP number (between 0 and 255) indicating which key was used (see the **LEVEL II COBOL Operating Guide** for details of defining function keys). Character 3 is undefined.

    9 An error has occurred. A COMP number in Character 2 determines the error as follows:

    0 - a null ACCEPT has occurred, for example a "clear screen" terminated the ACCEPT.
    8 - a character that has been disabled has been keyed.
    9 - an invalid keystroke (more than one byte) has occurred.

    Character 3 is undefined.

    Character 3 is used only in Japanese implementations.

Format 3

18. The ACCEPT statement causes the information requested to be transferred to the data item specified by identifier according to the rules of the MOVE statement. DATE, DAY, and TIME are conceptual data items and, therefore, are not described in the COBOL program.

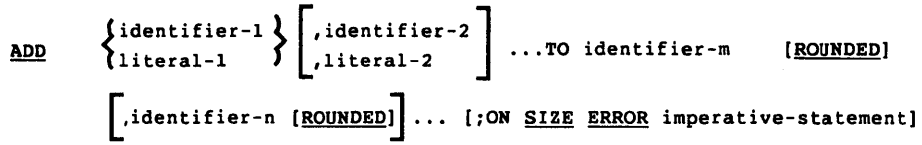
The ADD Statement

FUNCTION

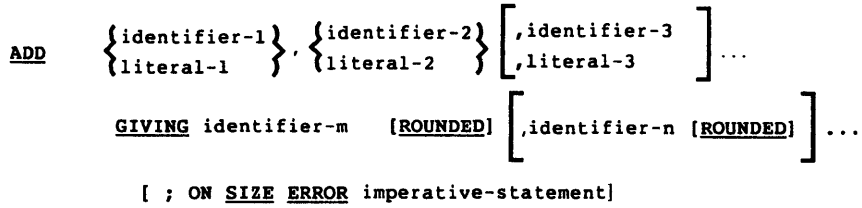
The ADD statement causes two or more numeric operands to be summed and the result to be stored.

GENERAL FORMAT

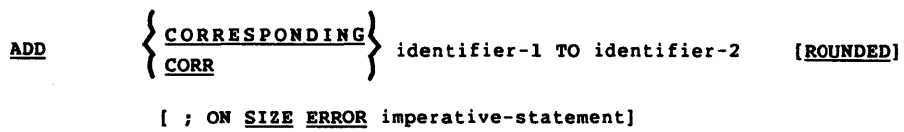
Format 1



Format 2



Format 3



The ALTER Statement

FUNCTION

The ALTER statement modifies a predetermined sequence of operations.

GENERAL FORMAT

```
ALTER procedure-name-1 TO   [PROCEED TO]   procedure-name-2

     [, procedure-name-3  TO  [PROCEED TO]  procedure-name-4  ] ...
```

SYNTAX RULES

1. Each procedure-name-1, procedure-name-3, ..., is the name of a paragraph that contains a single sentence consisting of a GO TO statement without the DEPENDING phrase.

2. Each procedure-name-2, procedure-name-4, ..., is the name of a paragraph or section in the Procedure Division.

GENERAL RULES

1. Execution of the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, procedure-name-3, ..., so that subsequent executions of the modified GO TO statements cause transfer of control to procedure-name-2, procedure-name-4, ..., respectively. Modified GO TO statements in independent segments may, under some circumstances, be returned to their initial states (see **Independent Segments** in Chapter 8).

2. A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different segment-number.

   All other uses of the ALTER statement are valid and are performed even if procedure-name-1, procedure-name-3 is in an overlayable fixed segment.

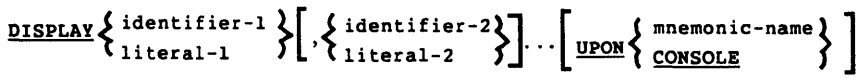
The DISPLAY Statement

FUNCTION

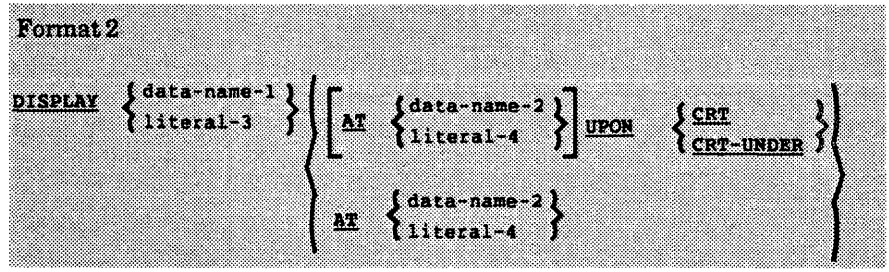
The DISPLAY statement causes data to be transferred from specified data items to the CRT screen.

GENERAL FORMATS

Format 1



Format 2



SYNTAX RULES

1. The mnemonic-name in Format-1 must be associated with the console in the SPECIAL-NAMES paragraph in the Environment Division.

2. Each literal may be any figurative constant, except ALL.

4. If the data item (or the portion of the data-item not yet transferred) is the same size as the data record, the data item (or the portion not yet transferred) is transferred.

5. If the data item (or the portion of the data item not yet transferred) is not the same size as the data record, one of the following applies:

   a. If the size of the data item (or the portion of the data item not yet transferred) exceeds the size of the data record, the data item (or the portion not yet transferred) is transferred to the data record, beginning with the leftmost character and continuing until the data record is filled, an additional data record is then requested.

   b. If the size of the data record exceeds the size of the data item (or the portion of the data item not yet transferred), the data item (or the portion not yet transferred) is transferred to the data record beginning with the leftmost character and continuing until the final character of the data item has been transferred. The remaining characters in the data record are space filled.

6. When operands in a DISPLAY statement are USAGE COMP or USAGE COMP-3 such operands are converted to USAGE DISPLAY. The size of the sending item is the sum of the sizes associated with the operands (after possible conversion) and the values of the operand are transferred in the sequence in which they are encountered.

Format 2

7. The DISPLAY statement is used to output data to the CRT in the screen positions specified.

8. data-name-1 is taken as a definition of the screen area into which data items that correspond to areas on the screen are moved. FILLER fields correspond to areas on the screen into which data is not moved.

9. data-name-1 may be an elementary item or may be a group item containing group and/or elementary items. Such elementary items must be defined as USAGE DISPLAY.

The DIVIDE Statement

FUNCTION

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient.

GENERAL FORMAT

Format 1

```
DIVIDE      {identifier-1}  INTO  identifier-2  [ROUNDED]
            {literal-1    }

    [, identifier-3 [ ROUNDED]] ... [;ON SIZE ERROR imperative-statement]
```

Format 2

```
DIVIDE    { identifier-1}  INTO    {identifier-2 }
          { literal-1   }          {literal-2    }

    GIVING identifier-3 [ROUNDED]      [, identifier-4 [ROUNDED]] ...

    [;ON SIZE ERROR imperative-statement]
```

Format 3

```
DIVIDE      {identifier-1}  BY    {identifier-2 }
            {literal-1    }       {literal-2    }

    GIVING  identifier-3  [ROUNDED]    [, identifier-4 [ROUNDED]] ...

    [;ON SIZE ERROR imperative-statement]
```

2. When Format 1 is used, the value of identifier-1 or literal-1 is divided into the value of identifier-2. The value of the dividend (identifier-2) is replaced by this quotient; similarly for identifier-1 or literal-1 and literal-3, etc.

3. When Format 2 is used, the value of identifier-1 or literal-1 is divided into identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4 etc.

4. When Format 3 is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4 etc.

5. Formats 4 and 5 are used when a remainder from the division operation is desired, namely identifier-4. The remainder in COBOL is defined as the result of subtracting the product of the quotient (identifier-3) and the divisor from the dividend. If identifier-3 is defined as a numeric edited item, the quotient used to calculate the remainder is an intermediate field which contains the unedited quotient. If ROUNDED is used, the quotient used to calculate the remainder is an intermediate field which contains the quotient of the DIVIDE statement, truncated rather than rounded.

6. In Formats 4 and 5, the accuracy of the REMAINDER data item (identifier-4) is defined by the calculation described above. Appropriate decimal alignment truncation (not rounding) will be performed for the content of the data item referenced by identifier-4, as needed.

7. When the ON SIZE ERROR phrase is used in Formats 4 and 5, the following rules pertain:

    a. If the size error occurs on the quotient, no remainder calculation is meaningful. Thus, the contents of the data items referenced by both identifier-3 and identifier-4 will remain unchanged.

    b. If the size error occurs on the remainder, the contents of the data item referenced by identifier-4 remains unchanged, However, as with other instances of multiple results of arithmetic statements, the user will have to do his own analysis to recognize which situation has actually occurred.

The EXIT Statement

FUNCTION

The EXIT statement provides a common end point for a series of procedures.

GENERAL FORMAT

```
EXIT.
```

SYNTAX RULES

1. The EXIT statement must appear in a sentence by itself.

2. The EXIT sentence must be the only sentence in the paragraph.

GENERAL RULE

An EXIT statement serves only to enable the user to assign a procedure-name to a given point in a program. Such an EXIT statement has no other effect on the compilation or execution of the program.

GENERAL RULES

1. When a GO TO statement, represented by Format 1 is executed, control is transferred to procedure-name-1 or to another procedure-name if the GO TO statement has been modified by an ALTER statement.

2. If procedure-name-1 is not specified in Format 1, an ALTER statement, referring to this GO TO statement, must be executed prior to the execution of this GO TO statement.

3. When a GO TO statement represented by Format 2 is executed, control is transferred to procedure-name-1, procedure-name-2, etc., depending on the value of the identifier being 1, 2, ..., *n*. If the value of the identifier is anything other than the positive or unsigned integers 1, 2, ..., *n*, then no transfer occurs and control passes to the next statement in the normal sequence for execution.

b. If the condition is true and the NEXT SENTENCE phrase is specified instead of statement-1, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

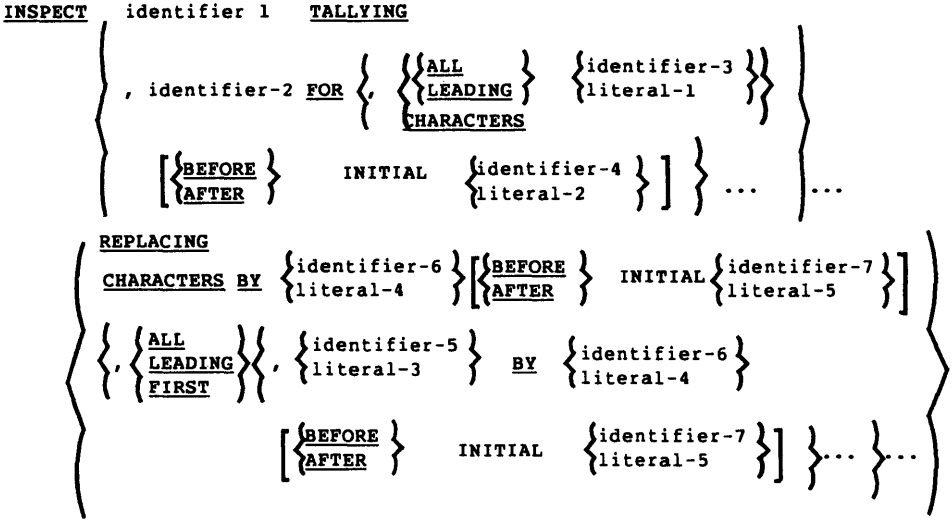
c. If the condition is false, statement-1 or its surrogate NEXT SENTENCE is ignored, and statement-2, if specified, is executed. If statement-2 contains a procedure branching or conditional statement, control is explicitly transferred in accordance with the rules of that statement. If statement-2 does not contain a procedure branching or conditional statement, control passes to the next executable sentence. If the ELSE statement-2 phrase is not specified, statement-1 is ignored and control passes to the next executable sentence.

d. If the condition is false, and the ELSE NEXT SENTENCE phrase is specified, statement-1 is ignored, if specified, and control passes to the next executable sentence.

2. statement-1 and/or statement-2 may contain an IF statement. In this case the IF statement is said to be nested.

   IF statements within IF statements may be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE.

Format 3



SYNTAX RULES

All Formats

1. identifier-1 must reference either a group item or any category of elementary item, described (either implicitly or explicitly) as USAGE IS DISPLAY.

2. identifier-3 ... identifier-n must reference either an elementary alphabetic, alphanumeric or numeric item described (either implicitly or explicitly) as USAGE IS DISPLAY.

3. Each literal must be nonnumeric and may be any figurative constant, except ALL.

4. literal-1, literal-2, literal-3, literal-4, and literal-5, and the data items referenced by identifier-3, identifier-4, identifier-5, identifier-6, and identifier-7 can be any number of characters in length up to the limit allowed for literals or data items.

b. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 is described as alphanumeric edited, numeric edited or unsigned numeric, the data item is inspected as though it had been redefined as alphanumeric (see general rule 2a) and the INSPECT statement had been written to reference the redefined data item.

c. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 is described as signed numeric, the data item is inspected as though it had been moved to an unsigned numeric data item of the same length and then the rules in general rule 2b had been applied. (See **The MOVE Statement** later in this chapter).

3. In general rules 4 through 11 all references to literal-1, literal-2, literal-3, literal-4, and literal-5 apply equally to the contents of the data item referenced by identifier-3, identifier-4, identifier-5, identifier-6, and identifier-7, respectively.

4. During inspection of the contents of the data item referenced by identifier-1, each properly matched occurrence of literal-1 is tallied (Formats 1 and 3) and/or each properly matched occurrence of literal-3 is replaced by literal-4 (Formats 2 and 3). *

5. The comparison operation to determine the occurrences of literal-1 to be tallied and/or occurrences of literal-3 to be replaced, occurs as follows:

   a. The operands of the TALLYING and REPLACING phrases are considered in the order they are specified in the INSPECT statement from left to right. The first literal-1, literal-3 is compared to an equal number of contiguous characters, starting with the leftmost character position in the data item referenced by identifier-1. literal-1, literal-3 and that portion of the contents of the data item referenced by identifier-1 match if, and only if, they are equal, character for character.

b. If the BEFORE phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1 from its leftmost character position up to, but not including, the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 5 is begun. If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase participates in the comparison operation as though the BEFORE phrase had not been specified.

c. If the AFTER phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase may participate only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1 from the character position immediately to the right of the rightmost character position of the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1 and the rightmost character position of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 5 is begun. If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase is never eligible to participate in the comparison operation.

Format 1

7. The contents of the data item referenced by identifier-2 is not initialized by the execution of the INSPECT statement.

Format 3

11. A Format 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 had been written with one statement being a Format 1 statement with TALLYING phrases identical to those specified in the Format 3 statement, and the other statement being a Format 2 statement with REPLACING phrases identical to those specified in the Format 3 statement. The general rules given for matching and counting apply to the Format 1 statement and the general rules given for matching and replacing apply to the Format 2 statement.

EXAMPLES:

Six examples of the use of the INSPECT statement follow:

INSPECT *word* TALLYING *count* FOR LEADING "L" BEFORE INITIAL "A", *count-1* FOR LEADING "A" BEFORE INITIAL "L".

Where *word* = LARGE, *count* = 1, *count-1* = 0.
Where *word* = ANALYST, *count* = 0, *count-1* = 1.

INSPECT *word* TALLYING *count* FOR ALL "L", REPLACING LEADING "A" BY "E" AFTER INITIAL "L".

Where *word* = CALLAR, *count* = 2, *word* = CALLAR.
Where *word* = SALAMI, *count* = 1, *word* = SALEMI.
Where *word* = LATTER, *count* = 1, *word* = LETTER.

INSPECT *word* REPLACING ALL "A" BY "G" BEFORE INITIAL "X".

Where *word* = ARXAX, *word* = GRXAX.
Where *word* = HANDAX, *word* = HGNDGX.

INSPECT *word* TALLYING *count* FOR CHARACTERS AFTER INITIAL "J" REPLACING ALL "A" BY "B".

The MOVE Statement

FUNCTION

The MOVE statement transfers data, in accordance with the rules of editing, to one or more data areas.

GENERAL FORMAT

Format 1

```
MOVE    { identifier-1 }    TO identifier-2[,identifier-3] ...
        { literal      }
```

Format 2

```
MOVE    { CORRESPONDING } identifer-1 TO identifier-2
        { CORR          }
```

SYNTAX RULES

1. identifier-1 and literal represent the sending area; identifier-2, identifier-3, ..., represent the receiving area.

2. CORR is an abreviation for CORRESPONDING.

3. When the CORRESPONDING phrase is used, both identifiers must be group items.

4. An index data item cannot appear as an operand of a MOVE statement. (See **The USAGE Clause** in this Chapter.)

The following rules apply to an elementary move between these categories:

a. i. The figurative constant SPACE, or an alphanumeric edited, or alphabetic data item must not be moved to a numeric or numeric edited data item.

ii. A numeric edited data item must not be moved to a numeric edited data item.

b. A numeric literal, the figurative constant ZERO, a numeric data item or a numeric edited data item must not be moved to an alphabetic data item.

c. A non-integer numeric literal or a non-integer numeric data item must not be moved to an alphanumeric or alphanumeric edited data item.

d. All other elementary moves are legal and are performed according to the rules given in general rule 4.

4. Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, along with any editing specified for the receiving data item:

a. When an alphanumeric edited or alphanumeric item is a receiving item, alignment and any necessary space filling takes place as defined under **Standard Alignment Rules** in this chapter. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled. If the sending item is described as being signed numeric, the operational sign will not be moved; if the operational sign occupies a separate character position (see **The SIGN Clause** in this chapter), that character will not be moved and the size of the sending item will be considered to be one less than its actual size (in terms of standard data format characters).

b. When a numeric or numeric edited item is the receiving item, alignment by decimal point and any necessary zero-filling takes place as defined under the **Standard Alignment Rules** in Chapter 2, except where zeros are replaced because of editing requirements.

Table 3-7. MOVE Statement Data Categories.

Category of Sending Data Item		Category of Receiving Data Item[1]			
		Alphabetic	Alphanumeric Edited Alphanumeric	Numeric Integer Numeric Non-Integer	Numeric Edited
ALPHABETIC		Yes/4c	Yes/4a	No/3a	No/3a
ALPHANUMERIC		Yes/4c	Yes/4a	Yes/4b	Yes/4b
ALPHANUMERIC EDITED		Yes/4c	Yes/4a	No/3a	No/3a
NUMERIC	INTEGER	No/2b	Yes/4a	Yes/4b	Yes/4b
	NON-INTEGER	No/3b	No/3c	Yes/4b	Yes/4b
NUMERIC EDITED		No/3b	Yes/4a	Yes/4b	No/3a

1 - The relevant rule number is quoted in these columns

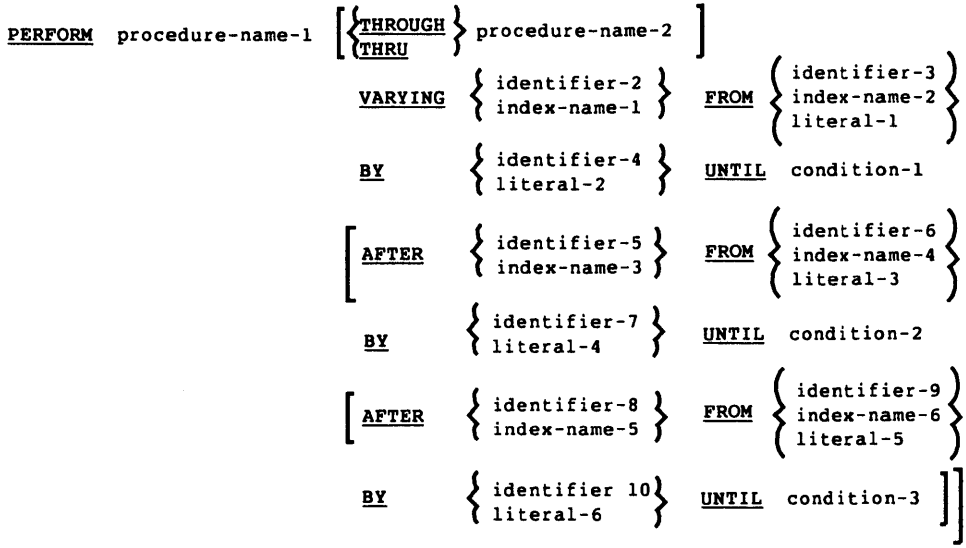
GENERAL RULES

1. See **The ROUNDED Phrase, The SIZE ERROR Phrase, Arithmetic Statements Overlapping Operands** and **Multiple Results in Arithmetic Statements** in this chapter.

2. When Format 1 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by this product; similarly for identifier-1 or literal-1 and identifier-3, etc.

3. When Format 2 is used, the value of identifier-1 or literal-1 is multiplied by identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4, etc.

Format 4



SYNTAX RULES

1. Each identifier represents a numeric elementary item described in the Data Division. In Format 2, identifier-1 must be described as a numeric integer.

2. Each literal represents a numeric literal.

3. The words THRU and THROUGH are equivalent.

4. If an index-name is specified in the VARYING or AFTER phrase, then:

   a. The identifier in the associated FROM and BY phrases must be an integer data item.

   b. The literal in the associated FROM phrase must be a positive integer.

   c. The literal in the associated BY phrase must be a non-zero integer.

b. If procedure-name-1 is a section-name and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.

c. If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

d. If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.

4. There is no necessary relationship between procedure-name-1 and procedure-name-2 except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. In particular, GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement, to which all of these paths must lead.

5. If control passes to these procedures other than via a PERFORM statement the procedures are executed right through to the next executable statement in the main program as if they were just part of the main program.

6. The PERFORM statements operate as follows with rule 5 above applying to all formats:

   a. Format 1 is the basic PERFORM statement. A procedure referenced by this type of PERFORM statement is excecuted once and then control passes to the next executable statement following the PERFORM statement.

   b. Format 2 is the PERFORM ...TIMES. The procedures are performed the number of times specified by integer-1 or by the initial value of the data item referenced by identifier-1 for the execution. If, at the time of execution of a PERFORM statement, the value of the data item referenced by identifier-1 is equal to zero or is negative, control passes to the next executable statement following the PERFORM statement. Following the execution of the procedures the specified number of times control is transferred to the next executable statement following the PERFORM statement.

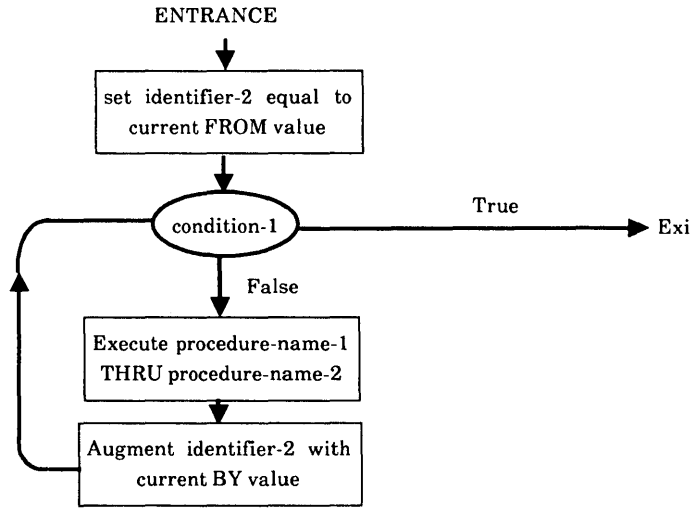


Figure 3-1. Flowchart of VARYING Phrase of a PERFORM Statement Having One Condition

In Format 4, when two identifiers are varied, identifier-2 and identifier-5 are set to the current value of identifier-3 and identifier-6, respectively. After the identifiers have been set, condition-1 is evaluated; if true, control is transferred to the next executable statement; if false, condition-2 is evaluated. If condition-2 is false, procedure-name-1 through procedure-name-2 is executed once, then identifier-5 is augmented by identifier-7 or literal-4 and condition-2 is evaluated again. This cycle of evaluation and augmentation continues until this condition is true. When condition-2 is true, identifier-5 is set to the value of literal-3 or the current value of identifier-4 and condition-1 is re-evaluated. The PERFORM statement is completed if condition-1 is true; if not, the cycles continue until condition-1 is true.

During the execution of the procedures associated with the PERFORM statement, any change to the VARYING variable (identifier-2 and index-name-1), the BY variable (identifier-4), the AFTER variable (identifier-5 and index-name-3), or the FROM variable (identifier-3 and index-name-2) will be taken into consideration and will affect the operation of the PERFORM statement.

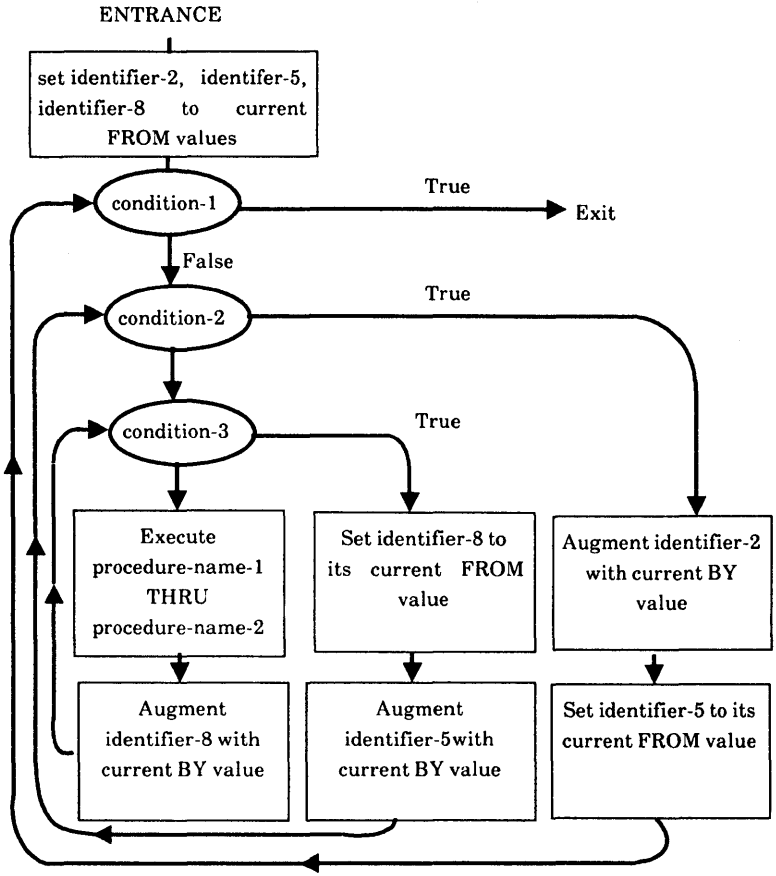


Figure 3-3. Flowchart for VARYING Phrase of PERFORM Statement with Three Conditions

After the completion of a Format 4 PERFORM statement, identifier-5 and identifier-8 contain the current value of identifier-6 and identifier-9 respectively. identifier-2 has a value that exceeds its last used setting by one increment or decrement value, unless condition-1 is true when the PERFORM statement is entered, in which case identifier-2 contains the current value of identifier-3.

8. A PERFORM statement that appears in a section that is not an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

   a. Sections and/or paragraphs wholly contained in one or more non-independent segments.

   b. Sections and/or paragraphs wholly contained in a single independent segment.

9. A PERFORM statement that appears in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

   a. Sections and/or paragraphs wholly contained in one or more non-independent segments.

   b. Sections and/or paragraphs wholly contained in the same independent segment as that PERFORM statement.

The STRING Statement

FUNCTION

The STRING statement provides juxtaposition of the partial or complete contents of two or more data items into a single data item.

GENERAL FORMAT

```
STRING { identifier-1 } [ , identifier-2 ] ... DELIMITED BY { identifier-3 }
       { literal-1    }   [ , literal-2    ]                 { literal-3    }
                                                             { SIZE         }

    [ , { identifier-4 } [ , identifier-5 ] ... ]
        { literal-4    }   [ , literal-5    ]

        DELIMITED BY { identifier-6 } ] ...
                     { literal-6    }
                     { SIZE         }

INTO   identifier-7   [WITH POINTER  identifier-8]

[, ON OVERFLOW imperative-statement]
```

SYNTAX RULES

1. Each literal may be any figurative constant without the optional word ALL.

2. All literals must be described as nonnumeric literals, and all identifiers, except identifier-8, must be described implicity or explicity as usage is DISPLAY.

3. identifier-7 must present an elementary alphanumeric data item without editing symbols or the JUSTIFIED clause.

4. identifier-8 must represent an elementary numeric integer data item of sufficient size to contain a value equal to the size plus 1 of the area referenced by identifier-7. The symbol 'P' may not be used in the PICTURE character-string of identifier-8.

c. If the DELIMITED phrase is specified with the SIZE phrase, the entire contents of literal-1, literal-2, or the contents of the data item referenced by identifier-1, identifier-2, are transferred, in the sequence specified in the STRING statement, to the data item referenced by identifier-7 until all data has been transferred or the end of the data item referenced by identifier-7 has been reached.

6. If the POINTER phrase is specified, identifier-8 is explicitly available to the programmer, who is then responsible for setting its initial value. The initial value must not be less than one.

7. If the POINTER phrase is not specified, the following general rules apply as if the user had specified identifier-8 with an initial value of 1.

8. When characters are transferred to the data item referenced by identifier-7, the moves behave as though the characters were moved one at a time from the source into the character position of the data item referenced by identifier-7 designated by the value associated with identifier-8, and then identifier-8 was increased by one prior to the move of the next character. The value associated with identifier-8 is changed during execution of the STRING statement only by the behaviour specified above.

9. At the end of execution of the STRING statement, only the portion of the data item referenced by identifier-7 that was referenced during the execution of the STRING statement is changed. All other portions of the data item referenced by identifier-7 will contain data that was present before this execution of the STRING statement.

10. If at any point at or after initialization of the STRING statement, but before execution of the STRING statement is completed, the value associated with identifier-8 is either less than one or exceeds the number of character positions in the data item referenced by identifier-7, no (further) data is transferred to the data item referenced by identifier-7, and the imperative statement in the ON OVERFLOW phrase is executed, if specified.

11. If the ON OVERFLOW phrase is not specified when the conditions described in general rule 10 above are encountered, control is transferred to the next executable statement.

2. Each literal must be a numeric literal.

3. The composite of operands must not contain more than 18 digits. (See **The Arithmetic Statements** in this Chapter).

   a. In Format 1 the composite of operands is determined by using all of the operands in a given statement.

   b. In Format 2 the composite of operands is determined by using all of the operands in a given statement excluding the data items that follow the word GIVING.

   c. In Format 3 the composite operands is determined separately for each pair of corresponding data items.

4. CORR is an abbreviation for CORRESPONDING.

GENERAL RULES

1. See **The ROUNDED Phrase, The SIZE ERROR Phrase, Arithmetic Statements, Overlapping Operands and Multiple Results in Arithmetic Statements** in this Chapter.

2. In Format 1, all literals or identifiers preceding the word FROM are added together and this total is subtracted from the current value of identifier-m storing the result immediately into identifier-m, and repeating this process respectively for each operand following the word FROM.

3. In Format 2, all literals or identifiers preceding the word FROM are added together, the sum is subtracted from literal-m or identifier-m and the result of the subtraction is stored as the new value of identifier-n, identifier-o, etc.

4. If Format 3 is used, data items in identifier-1 are subtracted from and stored into corresponding data items in identifier-2.

5. The compiler ensures enough places are carried so as not to lose significant digits during execution.

5. No identifier may name a level 88 entry.

6. The DELIMITER IN phrase and the COUNT IN phrase may be specified only if the DELIMITED BY phrase is specified.

GENERAL RULES

1. All references to identifier-2, literal-1, identifier-4, identifier-5 and identifier-6, apply equally to identifier-3, literal-2, identifier-7, identifier-8 and identifier-9, respectively, and all recursions thereof.

2. identifier-1 represents the sending area.

3. identifier-4 represents the data receiving area. identifier-5 represents the receiving area for delimiters.

4. literal-1 or the data item referenced by identifier-2 specifies a delimiter.

5. The data-item referenced by identifier-6 represents the count of the number of characters within the data item referenced by identifier-1 isolated by the delimiters for the move to the data-item referenced by identifier-4. This value does not include a count of the delimiter character(s).

6. The data item referenced by identifier-10 contains a value that indicates a relative character position within the area defined by identifier-1.

7. The data item referenced by identifier-11 is a counter that records the number of data items acted upon during the execution of an UNSTRING statement.

8. When a figurative constant is used as the delimiter, it stands for a single character nonnumeric literal.

   When the ALL phrase is specified, one occurrence or two or more contiguous occurrences of literal-1 (figurative constant or not) or the contents of the data item referenced by identifier-2 are treated as if it were only one occurrence, and this occurrence is moved to the receiving data item according to the rules in general rule 13d.

If the end of the data item referenced by identifier-1 is encountered before the delimiting condition is met, the examination terminates with the last character examined.

c. The characters thus examined (excluding the delimiting character(s), if any) are treated as an elementary alphanumeric data item, and are moved into the current receiving area according to the rules for the MOVE statement. (See **The MOVE Statement.**)

d. If the DELIMITER IN phrase is specified, the delimiting character(s) are treated as an elementary alphanumeric data item and are moved into the data item referenced by identifer-5 according to the rules for the MOVE statement. (see **The MOVE Statement.**) If the delimiting condition is the end of the data item referenced by identifier-1, then the data item referenced by identifier-5 is space filled.

e. If the COUNT IN phrase is specified, a value equal to the number of characters thus examined (excluding the delimiter character(s) if any) is moved into the area referenced by identifier-9 according to the rules for an elementary move.

f. If the DELIMITED BY phrase is specified the string of characters is further examined beginning with the first character to the right of the delimiter. If the DELIMITED BY phrase is not specified, the string of characters is further examined beginning with the character to the right of the last character transferred.

g. After data is transferred to the data item referenced by identifier-4, the current receiving area is the data item referenced by identifier-7. The behaviour described in paragraph 13b through 13f is repeated until either all the characters are exhausted in the data item referenced by identifier-1, or until there are no more receiving areas.

14. The initialization of the contents of the data items associated with the POINTER phrase or the TALLYING phrase is the responsibility of the user.

15. The contents of the data item referenced by identifier-10 will be incremented by one for each character examined in the data item referenced by identifier-1. When the execution of an UNSTRING statement with a POINTER phrase is complete, the contents of the data item referenced by identifier-10 will contain a value equal to the intial value plus the number of characters examined in the data item referenced by identifier-1.

CHAPTER 4

TABLE HANDLING

INTRODUCTION TO THE TABLE HANDLING MODULE

The Table Handling module provides a capability for defining tables of contiguous data items and accessing an item relative to its position in the table. Language facilities are provided for specifying how many times an item is to be repeated. Each item may be identified through use of a subscript or an index (see Chapter 2).

Table Handling provides a capability for accessing items in variable length tables of multiple dimensions. In ANSI standard COBOL the maximum number of multiple dimensions is three, but in LEVEL II COBOL it is 49. In practice, however, the maximum number of multiple dimensions is restricted by the maximum size of the Data Division. In addition table handling provides facilities for specifying ascending or descending keys and permits searching a dimension of a table for an item satisfying a specified condition.

DATA DIVISION IN THE TABLE HANDLING MODULE

The OCCURS Clause

FUNCTION

The OCCURS clause eliminates the need for separate entries for repeated data items and supplies information required for the application of subscripts or indices.

6. An INDEXED BY phrase is required if the subject of this entry, or an entry subordinate to this entry, is to be referred to by indexing. The index-name identified by this clause is not defined elsewhere, and not being data, cannot be associated with any data hierarchy. The compiler generates an implicit data-item with USAGE INDEX.

7. A data description entry that contains Format-2 of the OCCURS clause may only be followed, within that record description, by data description entries which are subordinate to it.

8. The OCCURS clause cannot be specified in a data description entry that:

   a. Has 01, 66, 77 or 88 level-number (ANSI standard COBOL only).

   b. Describes an item whose size is variable. The size of an item is variable if the data description of any subordinate item contains Format 2 of the OCCURS clause.

9. In Format 2, the data item defined by data-name-1 must not occupy a character position within the range of the first character position defined by the data description entry containing the OCCURS clause and the last character position defined by the record description entry containing that OCCURS clause.

10. If data-name-2 is not the subject of this entry, then:

    a. All of the items identified by the data-names in the KEY IS phrase must be within the group item which is the subject of this entry.

    b. Items identified by the data-name in the KEY IS phrase must not contain an OCCURS clause.

    c. There must not be any entry that contains an OCCURS clause between the items identified by the data-names in the KEY IS phrase and the subject of this entry.

11. index-name-1, index-name-2, ... must be unique words within the program.

5. The KEY IS phrase is used to indicate that the repeated data is arranged in ascending or descending order according to the values contained data-name-2, data-name-3, etc. The ascending or descending order is determined according to the rules for comparison of operands (see **Comparison of Numeric Operands, Comparison of Nonnumeric Operands** in Chapter 3). The data-names are listed in their descending order of significance.

The USAGE Clause

FUNCTION

The USAGE clause specifies the format of a data item in the computer storage.

GENERAL FORMAT

```
[USAGE IS]    INDEX
```

SYNTAX RULES

1. An index data item can be referenced explicitly only in a SEARCH or SET statement, a relation condition, the USING phrase of a Procedure Division header, or the USING phrase of a CALL statement.

2. The SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

GENERAL RULES

1. The USAGE clause can be written at any level. If the USAGE clause is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

The SEARCH Statement

FUNCTION

The SEARCH statement is used to search a table for a table element that satisfies the specified condition and to adjust the associated index-name to indicate that table element.

GENERAL FORMAT

Format 1

```
SEARCH  identifier-1     [VARYING  {identifier-2}]
                                   {index-name-1}

    [;  AT END  imperative-statement-1]

     ; WHEN condition-1   {imperative-statement-2}
                          {NEXT SENTENCE        }

   [ ; WHEN condition-2   {imperative-statement-3}  ]      ...
                          {NEXT SENTENCE        }
```

SYNTAX RULES

1. In both Formats 1 and 2, identifier-1 must not be subscripted or indexed, but its description must contain an OCCURS clause and an INDEXED BY clause. The description of identifier-1 in Format 2 must also contain the KEY IS phrase in its OCCURS clause.

2. identifier-2, when specified, must be described as USAGE IS INDEX or as a numeric elementary item without any positions to the right of the assumed decimal point.

3. In Format 1, condition-1, condition-2, etc., may be any condition as described in **Conditional Expressions** in Chapter 3.

4. In Format 2, all referenced condition names must be defined as having only a single value. The data-name associated with a condition-name must appear in the KEY clause of identifier-1. Each data-name-1, data-name-2 may be qualified. Each data-name-1, data-name-2 must be indexed by the first index-name associated with identifier-1 along with other indices or literal as required, and must be referenced in the KEY clause of identifier-1. identifier-3, identifier-4, or identifiers specified in arithmetic-expression-1, arithmetic-expression-2 must not be referenced in the KEY clause of identifier-1 or be indexed by the first index-name associated with identifier-1.

   In Format 2, when a data-name in the KEY clause of identifier-1 is referenced, or when a condition-name associated with a data-name in the KEY clause of identifier-1 is referenced, all preceding data-names in the KEY clause of identifier-1 or their associated condition-names must also be referenced.

GENERAL RULES

1. If Format 1 of the SEARCH is used, a serial type of search operation takes place, starting with the current index setting.

3. If Format 2 of the SEARCH is used, a non-serial operation may take place; the initial setting of the index-name for identifier-1 is ignored and its setting is varied during the search operation with the restriction that at no time is it set to a value that exceeds the value which corresponds to the last element of the table, or that is less than the value that corresponds to the last element of the table. The length of the table is discussed in the OCCURS clause. If any of the conditions specified in the WHEN clause cannot be satisfied for any setting of the index within the permitted range, control is passed to imperative-statement-1 of the AT END phrase, when specified, or to the next executable sentence when this phrase is not specified; in either case the final setting of the index is not predictable. If all conditions can be satisfied, the index indicates an occurrence that allows the condition to be satisfied, and control passes to imperative-statement-2.

4. After execution of imperative-statement-1, imperative-statement-2, or imperative-statement-3, that does not terminate with a GO TO statement, control passes to the next executable sentence.

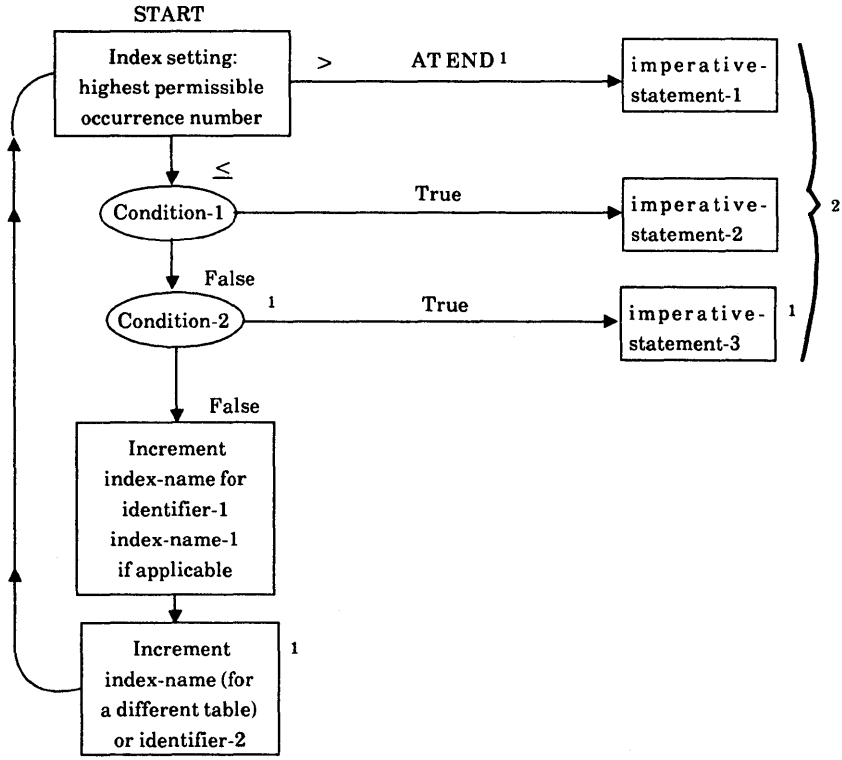
5. In Format 2, the index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY phrase of identifier-1. Any other index-names for identifier-1 remain unchanged.

6. In Format 1, if the VARYING phrase is not used, the index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY phrase of identifier-1. Any other index-names for identifier-1 remain unchanged.

7. In Format 1, if the VARYING index-name-1 phrase is specified, and if index-name-1 appears in the INDEXED BY phrase of identifier-1, that index-name is used for this search. If this is not the case, or if the VARYING identifier-2 phrase is specified, the first (or only) index-name given in the INDEXED BY phrase of identifier-1 is used for the search. In addition, the following operations will occur:

    a. If the VARYING index-name-1 phrase is used, and if index-name-1 appears in the INDEXED BY phrase of another table entry, the occurrence number represented by index-name-1 is incremented by the same amount as, and at the same time as, the index-name associated with identifier-1 is incremented.

Figure 4-1 shows a flowchart of the Format 1 SEARCH operation containing two WHEN phrases.



1 - These operations are options included only when specified in the SEARCH statement.

2 - Each of these control transfers is to the next executable sentence unless the imperative-statement ends with a GO TO statement.

Figure 4-1. Flowchart of SERCH Operation with Two WHEN Phrases.

GENERAL RULES

1. Index-names are considered related to a given table in ANSI standard COBOL and are defined by being specified in the INDEXED BY clause.

2. If index-name-3 is specified, the value of the index before the execution of the SET statement must correspond to an occurrence number of an element in the associated table.

   If index-name-4, index-name-5 is specified, the value of the index both before and after the execution of the SET statement must correspond to an occurrence number of an element in the associated table. If index-name-1, index-name-2 is specified, the value of the index after the execution of the SET statement must correspond to an occurrence number of an element in the associated table. The value of the index associated with an index-name after the execution of a SEARCH or PERFORM statement may be undefined. (See **The SEARCH Statement** and **The PERFORM Statement** in Chapter 3).

3. In Format 1, the following action occurs:

   a. index-name-1 is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element referenced by index-name-3, identifier-3 or integer-1. If identifier-3 is an index data item, or if index-name-3 is related to the same table as index-name-1, no conversion takes place.

   b. If identifier-1 is an index data item, it may be set equal to either the contents of index-name-3 or identifier-3 where identifier-3 is also an index item; no conversion takes place in either case.

   c. If identifier-1 is not an index data item, it may be set only to an occurrence number that corresponds to the value of index-name-3. Neither identifier-3 nor integer-1 can be used in this case.

   d. The process is repeated for index-name-2, identifier-2, etc., if specified. Each time the value of index-name-3 or identifier-3 is used as it was at the beginning of the execution of the statement. Any subscripting or indexing associated with identifier-1, etc., is evaluated immediately before the value of the respective data item is changed.

CHAPTER 5

SEQUENTIAL INPUT AND OUTPUT

INTRODUCTION TO THE SEQUENTIAL I-O MODULE

The sequential I-O module provides the capability of accessing records of a file in established sequence. The sequence is established as a result of writing the records to the file. It also provides for the specification of re-run points and the sharing of memory areas among files.

Language Concepts

ORGANIZATION

Sequential files are organized such that each record in the file except the first has a unique predecessor record, and each record except the last has a unique successor record. These predecessor-successor relationships are established by the order of WRITE statements when the file is created. Once established, the predecessor-successor relationships do not change except in the case where records are added to the end of the file.

ACCESS MODE

In the sequential access mode, the sequence in which records are accessed is the order in which the records were originally written.

3 - Permanent Error. The input-output statement was unsuccessfully executed as the result of a boundary violation for a sequential file or as the result of an input-output error, such as data check parity error, or transmission error.

9 - Operating System Error Message. The input-output statement was unsuccessfully executed as a result of a condition that is specified by the Operating System Error Message. This value is used only to indicate a condition not indicated by other defined values of status key 1, or by specified combinations of the values of status key 1 and status key 2.

Status Key 2

The rightmost character position of the FILE STATUS data item is known as Status Key 2 and is used to further describe the results of the input-output operation. This character may contain a value as follows:

- If no further information is available concerning the input-output operation, then status key 2 contains a value of '0'.
- When status key 1 contains a value of '3' an irrecoverable error has occurred. This is treated as a fatal error by the Operating System.
- When status key 1 contains a value of '9', the value of status key 2 is the operating system error message number (for those operating systems which designate errors numerically). The **LEVEL II COBOL Operating Guide** contains details of this status-key-2 representation.

  Note that it is not possible to extract this number directly.

*

Valid Combinations of Status Keys 1 and 2

The valid permissible combinations of the values of status key 1 and status key 2 are shown in the following table. An 'X' at an intersection indicates a valid permissible combination.

SHARING FILES ON MULTI-USER SYSTEMS

The Run-Time System (RTS) supports the LEVEL II COBOL multi-user facilities which allow data files to be shared between users in a multi-user environment, and allow programs accessing those files to prevent access to records or entire files while data is being updated.

Files are either active or inactive: an inactive file is one that is not open to any run unit, an active file is open to one or more run units.

Active files may be in one of two modes: exclusive or shareable.

Exclusive

A file which is in exclusive mode is open to one run unit only, and any other run unit which attempts to access it receives a "File locked" error and is denied access. Exclusive mode implies that a file lock is held by the one run unit which is able to access the file; the file lock is released by that run unit closing the file.

Shareable

A file which is in shareable mode is available to any number of run units, each of which may protect data while using the file by locking one record in the file. This prevents other run units accessing the individual records that are locked, but does not prevent access to the file otherwise. Line sequential files that are opened in INPUT or EXTEND mode can explicitly or implicitly by made shareable, but records cannot be locked in the file. A run unit cannot lock multiple records in a file whose organization is sequential. Each run unit that is sharing access to a sequential file may be locking a single record in that file.

SINGLE RECORD LOCK

A run unit that has specified single record locking for a file (either explicitly or implicitly) can hold only one record lock in that file at any time. There are two ways that a run unit can acquire a record lock: manually or automatically.

Note that:

- A file opened for OUTPUT causes the file to become exclusive, regardless of the specified lock mode.
- Explicitly or implicitly specifying automatic or manual record locking for a file causes the file to become shareable. A file opened for I-O acquires record locks, a file opened for INPUT or EXTEND never acquires record locks.
- The programmer can select the type of locking for individual files by accepting the default locking (see Table 5-1) or by including a LOCK MODE clause in the SELECT statement for a file (see **The SELECT Statement** later in this chapter).

In single user environments the multi-user syntax has no effect at run-time, but programs can be developed for use in both single and multi-user environments.

ENVIRONMENT DIVISION IN THE SEQUENTIAL I-O MODULE

Input-Output Section

THE FILE-CONTROL PARAGRAPH

Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

General Format

```
{FILE-CONTROL.}   file-control-entry-1 [file-control-entry-2]   ...
```

Syntax Rules

1. The SELECT clause must be specified first in the file control entry. The clauses which follow the SELECT clause may appear in any order.

2. Each file described in the Data Division must be named once and only once as file-name in the FILE-CONTROL paragraph. Each file specified in the file control entry must have a file description entry in the Data Division.

3. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause is implied.

4. data-name-1 must be defined in the Data Division as a two-character data item of the category alphanumeric and must not be defined in the File Section or the Communication Section.

5. data-name-1 may be qualified.

6. When the ORGANIZATION IS SEQUENTIAL clause is not specified, the ORGANIZATION IS SEQUENTIAL clause is implied.

7. The OPTIONAL phrase may only be specified for input files. Its specification is required for input files that are not necessarily present each time the object program is executed.

8. file-identifier is any user-defined word, but must not be the same as file-name.

9. **The NOT OPTIONAL phrase may be specified only for files to be opened input-output.**

6. When the FILE STATUS clause is specified, a value will be moved by the operating system into the data item specified by data-name-1 after the execution of every statement that references that file either explicitly or implicitly. This value indicates the status of execution of the statement (See **I-O Status** in this chapter.)

7. file-identifier will be implicitly defined if it is not explicitly defined.

8. Using the LINE ADVANCING FILE phrase causes a file suitable for a printer to be produced. This file will have no space compression to tab characters, there will be an initial carriage-return (0D hex) character and each record is written with AFTER ADVANCING 1 LINE as the default advancing phrase.

9. The PASSWORD IS data-name-2 phrase is for documentary purposes only. This syntax is valid only if the IBM compiler directive is specified at compile time.

10. The LOCK MODE clause is an optional clause of the file control entry, and is used to specify the locking technique used for the file.

    If this clause is omitted, opening the file causes it to become exclusive, unless the file is opened for INPUT. In this case, the file becomes shareable.

    When LOCK MODE IS EXCLUSIVE is specified, the run unit acquires a lock on the whole file when it opens the file.

    When LOCK MODE IS AUTOMATIC or LOCK MODE IS MANUAL is specified, the file that the run unit opens is shareable.

    The WITH LOCK ON RECORD clause is for documentary purposes only. LOCK MODE IS MANUAL and LOCK MODE IS AUTOMATIC imply single record locking.

    Line sequential files cannot be opened for I-O and so cannot acquire a record lock.

Syntax Rules

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

2. file-name-1 must be a sequentially organized file.

3. The END OF REEL/UNIT clause may only be used if file-name-2 is a sequentially organized file.

4. When either the integer-1 RECORDS clause or the integer-2 CLOCK-UNITS clause is specified, user-defined-name must be given in the RERUN clause.

5. More than one RERUN clause may be specified for a given file-name-2 subject to the following restrictions:

   a. When multiple integer-1 RECORD clauses are specified, no two of them can specify the same file-name-2.

   b. When multiple END OF REEL or END OF UNIT clauses are specified, no two of them may specify the same file-name-2.

6. The two forms of the SAME clause (SAME AREA, SAME RECORD AREA) are considered separately in the following:

   More than one SAME clause may be included in a program, however:

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

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

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

7. The files referenced in the SAME AREA or SAME RECORD AREA clause need not all have the same organization or access.

Record Description Structure

A record description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name if required, followed by a series of independent clauses as required. A record description has a hierarchical structure and therefore the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description is defined in **Concept Of Levels** in Chapter 2, while the elements allowed in a record description are shown in the **Data Description - Complete Entry Skeleton** in Chapter 3.

The File Description-Complete Entry Skeleton

FUNCTION

The file description furnishes information concerning the physical structure, identification, and record names pertaining to a given file.

The BLOCK CONTAINS Clause

FUNCTION

The BLOCK CONTAINS clause specifies the size of a physical record.

GENERAL FORMAT

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

GENERAL RULE

This clause is required for documentation purposes only.

The CODE-SET Clause

FUNCTION

The CODE-SET clause specifies the character code set used to represent data on the external media.

GENERAL FORMAT

```
CODE-SET IS  alphabet-name
```

GENERAL RULES

1. The presence of more than one data-name indicates that the file contains more than one type of data record. These records may be of differing sizes, different formats, etc. The order in which they are listed is not significant.

2. Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.

The LABEL RECORDS Clause

FUNCTION

The LABEL RECORDS clause specifies whether labels are present.

GENERAL FORMAT

```
LABEL  {RECORDS IS }  {STANDARD }
       {RECORDS ARE}  {OMITTED  }
```

SYNTAX RULES

This clause is required in every file description entry, in ANSI standard COBOL.

GENERAL RULES

This clause is used for documentation purposes only.

GENERAL RULES

1. The LINAGE clause provides a means for specifying the size of a logical page in terms of number of lines. The logical page size is the sum of the values referenced by each phrase except the FOOTING phrase. If the LINES AT TOP or LINES AT BOTTOM phrases are not specified, the values for these functions are zero. If the FOOTING phrase is not specified, the assumed value is equal to integer-1, or the contents of the data item referenced by data-name-1, whichever is specified.

   There is not necessarily any relationship between the size of the logical page and the size of a physical page.

2. The value of integer-1 or the data item referenced by data-name-1 specifies the number of lines that can be written and/or spaced on the logical page. The value must be greater than zero. That part of the logical page in which these lines can be written and/or spaced is called the page body.

3. The value of integer-3 or the data item referenced by data-name-3 specifies the number of lines that comprise the top margin on the logical page. The value may be zero.

4. The value of integer-4 or the data item referenced by data-name-4 specifies the number of lines that comprise the bottom margin on the logical page. The value may be zero.

5. The value of integer-2 or the data item referenced by data-name-2 specifies the line number within the page body at which the footing area begins. The value must be greater than zero and not greater than the value of integer-1 or the data item referenced by data-name-1.

   The footing area comprises the area of the logical page between the line represented by the value of integer-2 or the data item referenced by data-name-2 and the line represented by the value of integer-1 or the data item referenced by data-name-1, inclusive.

6. The value of integer-1, integer-3, and integer-4, if specified, will be used at the time the file is opened by the execution of an OPEN statement with the OUTPUT phrase, to specify the number of lines that comprise each of the indicated sections of a logical page. The value of integer-2, if specified, will be used at that time to define the footing area. These values are used for all logical pages written for the file during a given execution of the program.

- When the ADVANCING identifier-2 or integer phrase of the WRITE statement is specified, the LINAGE-COUNTER is incremented by integer or the value of the data item referenced by identifier-2.

- When the ADVANCING phrase of the WRITE statement is not specified, the LINAGE-COUNTER is incremented by the value one. (See **The WRITE Statement**).

- The value of LINAGE-COUNTER is automatically reset to one when the device is repositioned to the first line that can be written on for each of the succeeding logical pages. (See **The WRITE Statement**).

d. The value of LINAGE-COUNTER is automatically set to one at the time an OPEN statement is executed for the associated file.

10. Each logical page is contiguous to the next with no additional spacing provided.

The RECORD CONTAINS Clause

FUNCTION

The RECORD CONTAINS clause specifies the size of data records.

GENERAL FORMAT

```
RECORD CONTAINS  [ integer-1 TO ]  integer-2  CHARACTERS
```

GENERAL RULE

The size of each data record is completely defined within the record description entry, therefore this clause is never required. The RECORD CONTAINS clause is specified for documentation purposes only.

3. A figurative constant may be substituted in the format above wherever a literal is specified.

PROCEDURE DIVISION IN THE SEQUENTIAL I-O MODULE

The CLOSE Statement

FUNCTION

The CLOSE statement terminates the processing of reels/units and files, with optional rewind and/or lock or removal where applicable.

GENERAL FORMAT

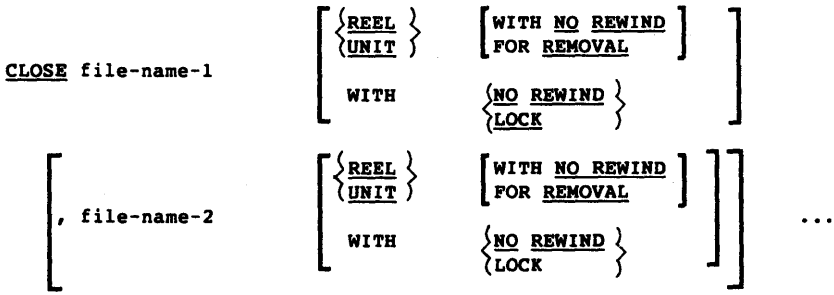


Table 5-2. Relationship of Categories of Files and the Formats of the CLOSE Statement

CLOSE Statement Format	File Category		
	Non-Reel/Unit	Sequential Single-Reel/Unit	Sequential Multi-Reel/Unit
CLOSE	C	C,G	C,G,A
CLOSE WITH LOCK	C,E	C,G,E	C,G,E,A
CLOSE WITH NO REWIND	X	C,B	C,B,A
CLOSE REEL/UNIT	X	X	F,G
CLOSE REEL/UNIT FOR REMOVAL	X	X	F,D,G
CLOSE REEL/UNIT WITH NO REWIND	X	X	F,B

The definitions of the symbols in the table are given below. Where the definition depends on whether the file is an input, output or input-output file, alternate definitions are given; otherwise, a definition applies to input, output, and input-output files.

A. Previous Reels/Units Unaffected

Input Files and Input-Output Files:

All reels/units in the file prior to the current reel/unit are processed except those reels/units controlled by a prior CLOSE REEL/UNIT statement. If the current reel/unit is not the last in the file, the reels/units in the file following the current one are not processed.

Output Files:

All reels/units in the file prior to the current reel/unit are processed except those reels/units controlled by a prior CLOSE REEL/UNIT statement.

F. Close Reel/Unit

Input Files:

The following operations take place:

1. A reel/unit swap.

2. The standard beginning reel/unit label procedure is executed.

The next executed READ statement for that file makes available the next data record on the new reel/unit.

Output Files and Input-Output Files:

The following operations take place;

1. (For output files only.) The standard ending reel/unit label procedure is executed.

2. A reel/unit swap.

3. The standard beginning reel/unit label procedure is executed.

For input-output files, the next executed READ statement that references that file makes the next logical data record on the next mass storage unit available. For output files, the next executed WRITE statement that references that file directs the next logical data record to the next reel/unit of the file.

G. Rewind

The current reel or analogous device is positioned at its physical beginning.

X. Illegal

This is an illegal combination of a CLOSE option and a file category. The results at object time are undefined.

The COMMIT Statement

FUNCTION

The COMMIT statement is specific to multi-user LEVEL II COBOL; it releases record locks.

GENERAL FORMAT

```
COMMIT
```

GENERAL RULES

Execution of the COMMIT statement causes the following action to occur:

- All record locks in all files held by the run unit are released.
- The file lock on each exclusive file is not affected by the COMMIT statement.

4. The EXTEND phrase must not be specified with multiple file reels.

5. The files referenced in the OPEN statement need not all have the same organization or access.

GENERAL RULES

1. The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode.

2. The successful execution of an OPEN statement makes the associated record area available to the program.

3. Prior to the successful execution of an OPEN statement for a given file, no statement (except for a SORT or MERGE statement with the USING or GIVING phrases) can be executed that references that file, either explicitly or implicitly.

4. An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statement. In Table 5-3, 'X' at an intersection indicates that the specified statement, used in the sequential access mode, may be used with the sequential file organization and open mode given at the top of the column.

Table 5-3. Permissible Combinations of Statements and OPEN Modes for Sequential I-O

Statement	Open Mode			
	Input	Output	Input-Output[1]	Extend
READ	X		X	
WRITE		X		X
REWRITE			X	

1 - This OPEN mode is not supported for ORGANIZATION line sequential files.

12. For files being opened with the INPUT or I-O phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. If no records exist in the file, the current record pointer is set such that the next executed READ statement for the file will result in an AT END condition. If the file does not exist, OPEN INPUT will cause an error status.

13. When the EXTEND phrase is specified, the OPEN statement positions the file immediately following the last logical record of that file. Subsequent WRITE statements referencing the file will add records to the file as though the file had been opened with the OUTPUT phrase. If the file does not exist it will be created.

14. When the EXTEND phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

    a. The beginning file labels are processed only in the case of a single reel/unit file.

    b. The beginning reel/unit labels on the last existing reel/unit are processed as though the file was being opened with the INPUT phrase.

    c. The existing ending file labels are processed as though the file is being opened with the INPUT phrase. These labels are then deleted.

    d. Processing then proceeds as though the file had been opened with the OUTPUT phrase.

15. The I-O phrase permits the opening of a file for both input and output operations except for files with ORGANIZATION LINE SEQUENTIAL. If the file does not exist it will be created and used as an empty file for input unless NOT OPTIONAL was specified in the SELECT statement. Any attempt to WRITE to it will cause an error.

16. When the I-O phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

    a. The labels are checked in accordance with the operating system specified conventions for input-output label checking.

The READ Statement

FUNCTION

The READ statement makes available the next logical record from a file.

GENERAL FORMAT

```
READ  filename  RECORD [INTO identifier]
          [WITH LOCK]  [; AT END imperative-statement]
```

SYNTAX RULES

1. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the record area associated with file-name must not be the same storage area.

2. The AT END phrase must be specified if no applicable USE procedure is specified for file-name.

GENERAL RULES

1. The associated file must be open in the INPUT or I-O mode at the time this statement is executed. (See **The OPEN Statement** in this chapter.)

2. The record to be made available by the READ statement is determined as follows:

   a. If the current record pointer was positioned by the execution of the OPEN statement, the record pointed to by the current record pointer is made available.

   b. If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file and then that record is made available.

10. If a file described with the OPTIONAL clause is not present at the time the file is opened, then at the time of the execution of the first READ statement for the file, the AT END condition occurs and the execution of the READ statement is unsuccessful. The standard end of file procedures are not performed. (See **The FILE-CONTROL Paragraph** and **The OPEN Statement** and **The USE Statement** descriptions in this chapter.) Execution of the program then proceeds as in general rule 12.

11. If, at the time of the execution of a READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful. (See **I-O Status.**)

12. When the AT END condition is recognized the following actions are taken in the specified order:

    a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition. (See **I-O Status.**)

    b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement. Any USE procedure specified for this file is not executed.

    c. If the AT END phrase is not specified, then a USE procedure must be specified, either explicitly or implicitly, for this file and that procedure is executed.

    When the AT END condition occurs, execution of the input-output statement which caused the condition is unsuccessful.

13. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined.

14. When the AT END condition has been recognized, a READ statement for that file must not be executed without first executing a successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

The REWRITE Statement

FUNCTION

The REWRITE statement logically replaces a record existing in a disk file.

GENERAL FORMAT

```
REWRITE record-name      [FROM identifier]
```

SYNTAX RULES

1. record-name and identifier must not refer to the same storage area.

2. record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

GENERAL RULES

1. The file associated with record-name must be a disk file and must be open in the I-O mode at the time of execution of this statement. (See **The OPEN Statement** in this chapter.)

2. The last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The operating system logically replaces the record that was accessed by the READ statement.

3. The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.

The UNLOCK Statement

FUNCTION

The UNLOCK statement releases all record locks acquired by the run unit on a named file.

GENERAL FORMAT

```
UNLOCK file-name.
```

GENERAL RULES

1. file-name must occur in the SELECT statement of the File Control entry.
2. The file referenced by file-name must already be opened with the OPEN statement.

GENERAL RULES

1. The designated procedures are executed by the input-output system after completing the standard input-output error routine; or upon recognition of the AT END condition, when the AT END phrase has not been specified in the input-output statement.

2. After execution of a USE procedure, control is returned to the invoking routine.

3. Within a USE procedure, there must not be any reference to any non-declarative procedures. Conversely, in the nondeclarative portion there must be no reference to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE statement or to the procedures associated with such a USE statement.

4. Within a USE procedure, there must not be the execution of any statement that would cause the execution of a USE procedure that had previously been invoked and had not yet returned control to the invoking routine.

6. If the END-OF-PAGE phrase is specified, the LINAGE clause must be specified in the file description entry for the associated file.

7. The words END-OF-PAGE and EOP are equivalent.

8. The ADVANCING TAB phrase cannot be specified when writing a record to a file whose file description entry contains the LINAGE clause.

GENERAL RULES

1. The associated file must be open in the OUTPUT or EXTEND mode at the time of the execution of this statement. (See **The OPEN Statement** in this chapter.)

2. The logical record released by the execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause or the execution of the WRITE statement was unsuccessful due to a boundary violation.

   The logical record is also available to the program as a record of other files referenced in the SAME RECORD AREA clause as the associated output file, as well as to the file associated with record-name.

3. The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of:

   a. The statement:

      MOVE identifier-1 TO record-name

      according to the rules specified for the MOVE statement, followed by:

   b. The same WRITE statement without the FROM phrase.

      The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.

iv. If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced according to the rules i, ii and iii above.

v. If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced according to the rules i, ii and iii above.

vi. If PAGE is specified, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page. If the record to be written is associated with a file whose file description entry contains a LINAGE clause, the repositioning is to the first line that can be written on the next logical page as specified in the LINAGE clause.

b. With ORGANIZATION LINE SEQUENTIAL, if the ADVANCING phrase is not used, automatic advancing of one line is provided to act in accordance with the convention of your operating system text editor (usually as if the user had specified BEFORE ADVANCING 1 LINE).

If the ADVANCING phrase is used, advancing is provided according to rules 9a(i) through 9a(vi) above.

10. If the logical end of the representation of the printed page is reached during the execution of a WRITE statement with the END-OF-PAGE phrase, the imperative-statement specified in the END-OF-PAGE phrase is executed. The logical end is specified in the LINAGE clause associated with record-name.

11. An end-of-page condition is reached whenever the execution of a given WRITE statement with the END-OF-PAGE phrase occurs when the execution of such a WRITE statement causes the LINAGE-COUNTER to equal or exceed the value specified by integer-2 or the data item referenced by data-name-2 of the LINAGE clause, if specified. In this case, the WRITE statement is executed and then the imperative statement in the END-OF-PAGE phrase is executed.

An automatic page overflow condition is reached whenever the execution of a given WRITE statement (with or without an END-OF-PAGE phrase) cannot be fully accommodated within the current page body.

CHAPTER 6

RELATIVE INPUT AND OUTPUT

INTRODUCTION TO THE RELATIVE I-O MODULE

The relative I-O module provides the capability of accessing records of a mass storage file in either a random or sequential manner. Each record in a relative file is uniquely identified by an integer value greater than zero which specifies the record's ordinal position in the file.

Language Concepts

ORGANIZATION

Relative file organization is permitted only on disk devices. A relative file consists of records which are identified by relative record numbers. The file may be thought of as composed of a serial string of areas, each capable of holding a logical record. Each of these areas is denominated by a relative record number. Record storage and retrieval is based on this number. For example, the tenth record is the one addressed by relative record number ten, and is in the tenth record area, whether or not records have been written in the first through the ninth record areas.

ACCESS MODES

In the sequential access mode, the sequence in which records are accessed is the ascending order of the relative record numbers of all records which currently exist within the file.

In the random access mode, the sequence in which records are accessed is controlled by the programmer. The desired record is accessed by placing its relative record number in a relative key data item.

1 - At End. The sequential READ statement was unsuccessfully executed either as a result of an attempt to read a record when no next logical record exists in the file or as a result of the first READ statement being executed for a file described with the OPTIONAL clause, and that file was not available to the program at the time its associated OPEN statement was executed.

3 - Permanent Error. The input-output statement was unsuccessfully executed as the result of a boundary violation for a sequential file or as the result of an input-output error, such as data check parity error, or transmission error.

9 - Operating System Error Message. The input-output statement was unsuccessfully executed as the result of a condition that is specified by the Operating System. This value is used only to indicate a condition not indicated by other defined values of status key 1, or by specified combinations of the values of status key 1 and status key 2.

Status Key 2

The rightmost character position of the FILE STATUS data item is known as status key 2 and is used to further describe the results of the input-output operation. This character contains a value as follows:

- If no further information is available concerning the input-output operation, then status key 2 contains a value of '0'
- When status key 1 contains a value of '2' indicating an INVALID KEY condition, status key 2 is used to designate the cause of that condition by the following values:

  2 - Indicates a duplicate key value. An attempt has been made to write a record that would create a duplicate key in a relative file.

  3 - Indicates no record found. An attempt has been made to access a record, identified by a key, and that record does not exist in the file.

  4 - Indicates a boundary violation. An attempt has been made to write beyond the externally-defined boundaries of a relative file. This is normally treated as a fatal error by the operating system.

When the INVALID KEY condition is recognized, the operating system takes these actions in the following order:

1. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an INVALID KEY condition. (See **I-O Status** in this chapter.)

2. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any USE procedure specified for this file is not executed.

3. If the INVALID KEY phrase is not specified, but a USE procedure is specified, either explicitly or implicitly, for this file, that procedure is executed.

When the INVALID KEY condition occurs, execution of the input-output statement which recognized the condition is unsuccessful, and the file is not affected.

NOTE:

> INVALID KEY does not trap errors when Status key 1 is set to 9. Such errors must be trapped either by explicitly testing the Status key or by using declaratives instead of the INVALID KEY clause.

THE AT END CONDITION

The AT END condition can occur as a result of the execution of a READ statement. For details of the causes of the condition, see **The READ Statement** later in this chapter.

Manual Record Locking

The run unit acquires a lock only if it accesses a record with a READ WITH LOCK statement. The lock is released by the same run unit a) accessing any record in the file with any file operation except START, b) executing an UNLOCK statement on that file, c) executing a COMMIT statement or d) closing the file.

Automatic Record Locking

The run unit acquires a lock whenever it reads a record in the file. The lock is released by the same run unit a) accessing any record in the same file with any file operation except START, b) executing an UNLOCK statement on that file, c) issuing a COMMIT statement or d) closing the file.

MULTIPLE RECORD LOCKS

A run unit that has specified multiple record locking for a file may hold a number of record locks in one file simultaneously. This prevents other run units accessing those locked records, but does not deny them access to any records that are not locked. There are two ways record locks can be acquired: manually or automatically.

Manual Record Locking

The run unit acquires a lock only if it accesses a record with a READ WITH KEPT LOCK statement. If the WRITELOCK compiler directive has been specified at the time the program was compiled then a lock is acquired if the run unit accesses the file with a WRITE or REWRITE statement. The locks are released by the same run unit a) executing an UNLOCK statement for that file, b) executing a COMMIT statement or c) closing the file.

Note that:

- A file opened for OUTPUT causes the file to become exclusive, regardless of the specified lock mode.
- Explicitly or implicitly specifying automatic or manual record locking for a file causes the file to become shareable. A file opened for I-O acquires record locks, a file opened for INPUT never acquires record locks.
- The programmer can select the type of locking for individual files by accepting the default locking (see Table 6-1) or by including a LOCK MODE clause in the SELECT statement for a file (see **The SELECT Statement** later in this chapter).

In single user environments the multi-user syntax has no effect at run-time but programs can be developed for use in both single and multi-user environments.

ENVIRONMENT DIVISION IN THE RELATIVE I-O MODULE

Input-output Section

THE FILE-CONTROL PARAGRAPH

Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information. (See also the **LEVEL II COBOL Operating Guide.**)

General Format

```
FILE-CONTROL   file-control-entry-1   [file-control-entry-2]   ...
```

3. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause is implied.

4. data-name-2 must be defined in the Data Division as a two-character data item of the category alphanumeric and must not be defined in the File Section or the Communication Section.

5. data-name-1 and data-name-2 may be qualified.

6. If a relative file is to be referenced by a START statement, the RELATIVE KEY phrase must be specified for that file.

7. data-name-1 must not be defined in a record description entry associated with that file-name.

8. The data item referenced by data-name-1 must be defined as an unsigned integer.

9. file-identifier is any user-defined word, but must not be the same as the file-name.

10. The NOT OPTIONAL phrase may be specified only for files opened input-output.

General Rules

1. The ASSIGN clause specifies the association of the file referenced by file-name to a storage medium. See the **LEVEL II COBOL Operating Guide.** The first assignment takes effect. Subsequent assignments within any one ASSIGN clause are for documentation purposes only.

2. The RESERVE clause allows the user to specify the number of input-output areas allocated. The RESERVE clause is treated as for documentation purposes only, unless the **LEVEL II COBOL Operating Guide** specific to your operating system indicates otherwise; it does, however, have the effect of implicitly declaring a data item AREA-VALUE, which is PIC 9(2) COMP and which contains the number of areas reserved.

When LOCK MODE IS EXCLUSIVE is specified, the run unit acquires a lock on the whole file when it opens the file.

When LOCK MODE IS AUTOMATIC or LOCK MODE IS MANUAL is specified, the file that the run unit opens is shareable.

13. The WITH LOCK ON RECORD clause specifies single record locking for the file.

The WITH LOCK ON MULTIPLE RECORDS clause specifies multiple record locking for the file. This clause must be present if multiple record locking is required.

If LOCK MODE IS AUTOMATIC WITH LOCK ON RECORD is specified for a file a record lock is acquired by the execution of the READ statement.

If LOCK MODE IS AUTOMATIC WITH LOCK ON MULTIPLE RECORDS is specified for a file, a record lock is acquired by the READ statement.

If LOCK MODE IS MANUAL WITH LOCK ON RECORD is specified a lock is acquired by a READ statement only if the READ statement includes the WITH LOCK phrase.

If LOCK MODE IS MANUAL WITH LOCK ON MULTIPLE RECORDS is specified, a record lock is acquired by the READ WITH KEPT LOCK statement.

When the lock mode is MANUAL or AUTOMATIC, single record locking is assumed unless the WITH LOCK ON MULTIPLE RECORDS phrase is included.

6. The two forms of the SAME clause (SAME AREA, SAME RECORD AREA) are considered separately in the following:

   More than one SAME clause may be included in a program, however:

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

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

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

7. The files referenced in the SAME AREA or SAME RECORD AREA clauses need not all have the same organization or access.

General Rules

1. The RERUN clause is treated as for documentation purposes only.

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

The File Description - Complete Entry Skeleton

FUNCTION

The file description furnishes information concerning the physical structure, identification, and record names pertaining to a given file.

GENERAL FORMAT

```
FD file-name

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

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

     [ ; LABEL { RECORD IS  } {STANDARD } ]
               { RECORDS ARE} {OMITTED  }

     [ ; VALUE OF data-name-1 IS {data-name-2}
                                 {literal-1  }
               [ , data-name-3 IS {data-name-4}  ]... ]
                                  {literal-2  }

     [ ; DATA  { RECORD IS  }  data-name-5 [,data-name-6]... ]
               { RECORDS ARE}
```

SYNTAX RULES

1. The level indicator FD identifies the beginning of a file description and must precede the file-name.

2. The clauses which follow the name of the file are optional in many cases, and their order of appearance is immaterial. All clauses are optional in LEVEL II COBOL, including the LABEL clause, which is required in ANSI standard COBOL.

3. One or more record description entries must follow the file description entry.

SYNTAX RULE

data-name-1 and data-name-2 are the names of data records and should have 01 level-number record descriptions, with the same names, associated with them.

GENERAL RULES

1. The presence of more than one data-name indicates that the file contains more than one type of data record. These records may be of differing sizes, different formats, etc. The order in which they are listed is not significant.

2. Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.

The LABEL RECORDS Clause

FUNCTION

The LABEL RECORDS clause specifies whether labels are present.

GENERAL FORMAT

```
LABEL   {RECORD IS  }{STANDARD}
        {RECORDS ARE}{OMITTED }
```

SYNTAX RULE

This clause is required in every file description entry, in ANSI standard COBOL.

The VALUE OF Clause

FUNCTION

The VALUE OF clause particularizes the description of an item in the label records associated with a file.

GENERAL FORMAT

```
VALUE OF data-name-1 IS        {data-name-2}
                               {literal-1  }

        [,data-name-3 IS       {data-name-4}]  ...
        [                      {literal-2  }]
```

SYNTAX RULES

1. Data-names should be qualified when necessary, but cannot be subscripted or indexed, nor can they be items described with the USAGE IS INDEX clause

2. data-name-2, data-name-4 etc, must be in the Working-Storage Section

GENERAL RULES

1. This clause is for documentation purposes only.

   The compiler checks that data-name-1 matches in value data-name-2 or literal-1, data-name-3 matches in value data-name-4 or literal-2, etc, for input files. For output files the value of data-name-2 or literal-1 is substituted for data-name-1, the value of data-name-4 or literal-2 is substituted for data-name-3, etc.

2. A figurative constant may be substituted in the format above wherever a literal is specified.

The definitions of the symbols in the table are given below. Where the definition depends on whether the file is an input, output or input-output file, alternate definitions are given; otherwise, a definition applies to input, output, and input-output files.

A. Close File

Input Files and Input-Output Files (Sequential Access Mode):

If the file is positioned at its end and label records are specified for the file, the labels are processed according to the operating system label convention. The behaviour of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. If the file is positioned at its end and label records are not specified for the file, label processing does not take place but other closing operations dependent on the Run-Time System (RTS) are executed. See your **LEVEL II COBOL Operating Guide**. If the file is positioned other than at its end, the closing operations dependent on the RTS are executed, but there is no ending label processing.

Input Files and Input-Output Files (Random or Dynamic Access Mode);
Output Files (Random, Dynamic, or Sequential Access Mode):

If label records are specified for the file, the labels are processed according to the operating system standard label convention. The behaviour of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. If label records are not specified for the file, label processing does not take place but other closing operations dependent on the RTS are executed.

B. File Lock

This file cannot be opened again during this execution of this run unit.

3. The action taken if a file is in the open mode when a STOP RUN statement is executed is to close the file. The action taken for a file that has been opened in a called program and not closed in that program prior to the execution of a CANCEL statement for the program is to close the file.

The COMMIT Statement

FUNCTION

The COMMIT statement is specific to multi-user LEVEL II COBOL; it releases record locks.

GENERAL FORMAT

```
COMMIT
```

GENERAL RULES

Execution of the COMMIT statement causes the following action to occur:

- All record locks in all files held by the run unit are released.
- The file lock on each exclusive file is not affected by the COMMIT statement.

4. After the succesful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.

5. The execution of a DELETE statement does not affect the contents of the record area associated with file-name.

6. The current record pointer is not affected by the execution of a DELETE statement.

7. The execution of the DELETE statement causes the value of the specified FILE STATUS data item, if any, associated with the file-name to be updated. See **I-O Status** in this chapter.

8. When using DELETE, the record to be deleted must not be locked by another run unit.

9. Following the successful execution of a DELETE statement, any record lock held by the run unit on the deleted record is released.

Table 6-2. Permissible Combinations of Statements and OPEN Modes for Relative I-O

File Access Mode	Statement	Open Mode		
		Input	Output	Input-Output
Sequential	READ	X		X
	WRITE		X	
	REWRITE			X
	START	X		X
	DELETE			X
Random	READ	X		X
	WRITE		X	X
	REWRITE			X
	START			
	DELETE			X
Dynamic	READ	X		X
	WRITE		X	X
	REWRITE			X
	START	X		X
	DELETE			X

5. A file may be opened with the INPUT, OUTPUT, and I-O phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent execution for that same file must be preceded by the execution of a CLOSE statement for that file.

6. Execution of the OPEN statement does not obtain or release the first data record.

7. The ASSIGNed name in the SELECT statement for a file is processed as follows:

   a. When the INPUT phrase is specified, the execution of the OPEN statement causes the ASSIGNed name to be checked in accordance with the operating system conventions for opening files for input.

14. When LOCK MODE IS EXCLUSIVE is specified, successful execution of an OPEN statement locks the file exclusively to that run unit.

15. When LOCK MODE IS AUTOMATIC or LOCK MODE IS MANUAL is specified, the file that is referred to is shareable. More than one run unit may successfully open such a file.

16. A file opened for OUTPUT is implicitly defined as a file with an exclusive lock, that is, it is not shareable.

17. Only shareable files opened for I-O can acquire record locks.

SYNTAX RULES

1. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the record area associated with file-name must not be the same storage area.

2. Format 1 must be used for all files in sequential access mode. *

3. Format 1 (with the NEXT phrase) must be specified for files in dynamic access mode, when records are to be retrieved sequentially.

4. Format 2 or Format 4 is used for files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.

5. The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file-name.

6. Format 1 or Format 2 must be used when single records are being locked in a shareable file.

7. Format 3 or Format 4 must be used when multiple records are being locked in a shareable file.

GENERAL RULES

1. The associated files must be open in the INPUT or I-O mode at the time this statement is executed. See **The OPEN Statement** in this chapter

2. The record to be made available by a Format 1 or Format 3 READ statement is determined as follows:

   a. The record, pointed to by the current record pointer, is made available provided that the current record pointer was positioned by the START or OPEN statement and the record is still accessible through the path indicated by the current record pointer; if the record is no longer accessible, which may have been caused by the deletion of the record, the current record pointer is updated to point to the next existing record in the file and that record is then made available.

10. When the AT END condition is recognized the following actions are taken in the specified order:

    a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition. (See **I-O Status** in this chapter.)

    b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement. Any USE procedure specified for this file is not executed.

    c. If the AT END phrase is not specified, then a USE procedure must be specified, either explicitly or implicitly, for this file, and that procedure is executed.

    When the AT END condition occurs, execution of the input-output statement which caused the condition is unsuccessful.

11. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined.

12. When the AT END condition has been recognized, a Format 1 or Format 3 READ statement for that file must not be executed without first executing one of the following:

    a. A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

    b. A successful START statement for that file.

    c. A successful Format 2 or Format 4 READ statement for that file.

13. For a file for which dynamic access mode is specified, a Format 1 or Format 3 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from the file as described in general rule 2.

14. If the RELATIVE KEY phrase is specified, the execution of a Format 1 or Format 3 READ statement updates the contents of the RELATIVE KEY data item such that it contains the relative record number of the record made available.

The REWRITE Statement

FUNCTION

The REWRITE statement logically replaces a record existing in a disk file.

GENERAL FORMAT

```
REWRITE record-name [FROM identifier][;INVALID KEY imperative-statement]
```

SYNTAX RULES

1. record-name and identifier must not refer to the same storage area.

2. record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

3. The INVALID KEY phrase must not be specified for a REWRITE statement which references a file in sequential access mode.

4. The INVALID KEY phrase must be specified in the REWRITE statement for files in the random or dynamic access mode for which an appropriate USE procedure is not specified.

GENERAL RULES

1. The file associated with record-name must be open in the I-O mode at the time of execution of this statement. (See **The OPEN Statement** in this chapter.)

2. For files in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The operating system logically replaces the record that was accessed by the READ statement.

The START Statement

FUNCTION

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

GENERAL FORMAT

```
START  file-name [ KEY  { IS EQUAL TO                } data-name ]
                        { IS =                       }
                        { IS GREATER THAN            }
                        { IS >                       }
                        { IS NOT LESS THAN           }
                        { IS NOT <                   }
     [;INVALID  KEY  imperative-statement]
```

NOTE:

The required relational characters '>', and '<' and '=' are not underlined to avoid confusion with other symbols such as '≥' (greater than or equal to).

SYNTAX RULES

1. file-name must be the name of a file with sequential or dynamic access.

2. data-name may be qualified.

3. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.

4. data-name, if specified, must be the data item specified in the RELATIVE KEY phrase of the associated file control entry.

GENERAL RULES

1. file-name must be open in the INPUT or I-O mode at the time that the START statement is executed. (See **The OPEN Statement** in this chapter.)

The UNLOCK Statement

FUNCTION

The UNLOCK statement releases all record locks acquired by the run unit on a named file.

GENERAL FORMAT

```
UNLOCK file-name.
```

GENERAL RULES

1. file-name must occur in the SELECT statement of the File Control entry.
2. The file referenced by file-name must already be opened with the OPEN statement.

GENERAL RULES

1. The designated procedures are executed by the input-output system after completing the standard input-output error routine, or upon recognition of the INVALID KEY or AT END conditions when the INVALID KEY or AT END phrases have not been specified in the input-output statement.

2. After execution of a USE procedure, control is returned to the invoking routine.

3. Within a USE procedure, there must not be any reference to any nondeclarative procedures. Conversely, in the nondeclarative portion there must be no reference to procedure-names in the declarative portion, except that PERFORM statements may refer to a USE statement or to the procedures associated with such a USE statement.

4. Within a USE procedure, there must not be the execution of any statement that would cause the execution of a USE procedure that had previously been invoked and had not yet returned control to the invoking routine.

3. The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of:

   a. The statement:

      MOVE identifier TO record-name

      according to the rules specified for the MOVE statement, followed by:

   b. The same WRITE statement without the FROM phrase.

      The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.

      After execution of the WRITE statement is complete, the information in the area referenced by identifier is available, even though the information in the area referenced by record-name may not be. (See general rule 2 above.)

4. The current record pointer is unaffected by the execution of a WRITE statement.

5. The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. (See **I-O Status** in this chapter.)

6. The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

7. The number of character positions on a mass storage device required to store a logical record in a file may or may not be equal to the number of character positions defined by the logical description of that record in the program.

8. The execution of the WRITE statement releases a logical record to the operating system.

13. If the file referred to has multiple record locking the WRITE statement can be made to acquire a record lock by use of the relevant compiler directives. Refer to the **LEVEL II COBOL Operating Guide** for details of these directives.

CHAPTER 7

INDEXED INPUT AND OUTPUT

INTRODUCTION TO THE INDEXED I-O MODULE

The indexed I-O module provides the capability of accessing records of a mass storage file in either a random or sequential manner. Each record in an indexed file is uniquely identified by the value of one or more keys within that record.

Language Concepts

ORGANIZATION

A file whose organization is indexed is a mass storage file in which data records may be accessed by the value of a key. A record description may include one or more key data items, each of which is associated with an index. Each index provides a logical path to the data records according to the contents of a data item within each record which is the record key for that index.

The data item named in the RECORD KEY clause of the file control entry for a file is the prime record key for that file. For purposes of inserting, updating and deleting records in a file, each record is identified solely by the value of its prime record key. This value must, therefore, be unique and must not be changed when updating the record. Key lengths depend on your operating system, and may be further restricted depending on your Indexed Sequential file run time module; see your LEVEL II COBOL Operating Guide.

A data item named in the ALTERNATE RECORD KEY clause of the file control entry for a file is an alternative record key for that file. The value of an alternative record key may be non-unique if the DUPLICATES phrase is specified for it. These keys provide alternative access paths for retrieval of records from the file. A maximum number of 80 alternate keys can be specified.

Status Key 1

The leftmost character position of the FILE STATUS data item is known as status key 1 and is set to indicate one of the following conditions upon completion of the input-output operation.

'0' - Successful Completion
'1' - At End
'2' - Invalid Key
'3' - Permanent Error
'9' - Operating System Error Message

The meaning of the above indications are as follows:

0 - Successful Completion. The input-output statement was successfully executed.

1 - At End. The Format 1 READ statement was unsuccessfully executed as a result of an attempt to read a record when no next logical record exists in the file.

2 - Invalid Key. The input-output statement was unsuccessfully executed as a result of one of the following:

Sequence Error
Duplicate Key
No Record Found
Boundary Violation

3 - Permanent Error. The input-output statement was unsuccessful as the result of an input-output error, such as data check, parity error, or transmission error.

9 - Operating System Error Message. The input-output statement was unsuccessfully executed as a result of a condition that is specified by the Operating System Error Message number. This value is used only to indicate a condition not indicated by other defined values of status key 1, or by specified combinations of the value of status key 1 and status key 2.

- When status key 1 contains a value of '9' the value of status key 2 is the operating system error message number (for those operating systems which designate errors numerically). The **LEVEL II COBOL Operating Guide** specific to your operating system contains details of the status-key-2 representation.

  Note that it is not possible to extract this number directly.

Valid Combinations of Status Keys 1 and 2

The valid permissible combinations of the value of status key 1 and status key 2 are shown in the following table. An 'X' at an intersection indicates a valid permissible combination.

Status Key 1	Status Key 2				
	No Further Information (0)	Sequence Error (1)	Duplicate Key (2)	No Record Found (3)	Boundary Violation (4)
Successful Completion (0)	X		X		
At End (1)	X				
Invalid Key (2)		X	X	X	X
Permanent Error (3)	X				
Implementor Defined (9) Operating System Error Message Number					

*

SHARING FILES ON MULTI-USER SYSTEMS

The Run-Time System (RTS) supports the LEVEL II COBOL multi-user facilities which allow data files to be shared between users in a multi-user environment, and allow programs accessing those files to prevent access to records or entire files while data is being updated.

Files are either active or inactive: an inactive file is one that is not open to any run unit, an active file is open to one or more run units.

Active files may be in one of two modes: exclusive or shareable.

Exclusive

A file which is in exclusive mode is open to one run unit only, and any other run unit which attempts to access it receives a "File locked" error and is denied access. Exclusive mode implies that a file lock is held by the one run unit which is able to access the file; the file lock is released by that run unit closing the file.

Shareable

A file which is in shareable mode is available to any number of run units, each of which may protect data while using the file by locking one or more records in the file. This prevents other run units accessing the individual records that are locked, but does not prevent access to the file otherwise. Files that are opened in INPUT mode are shareable, but records cannot be locked in the file. Each run unit that is sharing access to file a may be locking either a single record or multiple records in that file. A run unit cannot select single record locking and multiple record locking for the same file.

SINGLE RECORD LOCK

A run unit that has specified single record locking for a file (either explicitly or implicitly) can hold only one record lock in that file at any time. There are two ways that a run unit can acquire a record lock: manually or automatically.

Automatic Record Locking

The run unit acquires a lock whenever it reads a record in the file. If the WRITELOCK compiler directive has been specified at the time the program was compiled then a lock is acquired if the run unit accesses the file with a WRITE or REWRITE statement. The locks are released by the same run unit a) executing an UNLOCK statement for that file, b) executing a COMMIT statement or c) closing the file.

Table 7-1 shows the default type of locking which is used when files are opened in a particular open mode. The default locking can be modified if the AUTOLOCK compiler directive was specified at the time the program was compiled. The table also indicates whether the default type of locking may be overridden for individual files. This is done by inserting a suitable clause in the SELECT statement for the file; refer to **The SELECT Statement** later in this chapter for details of the required syntax.

Table 7-1. Default Locking for Indexed Data Files

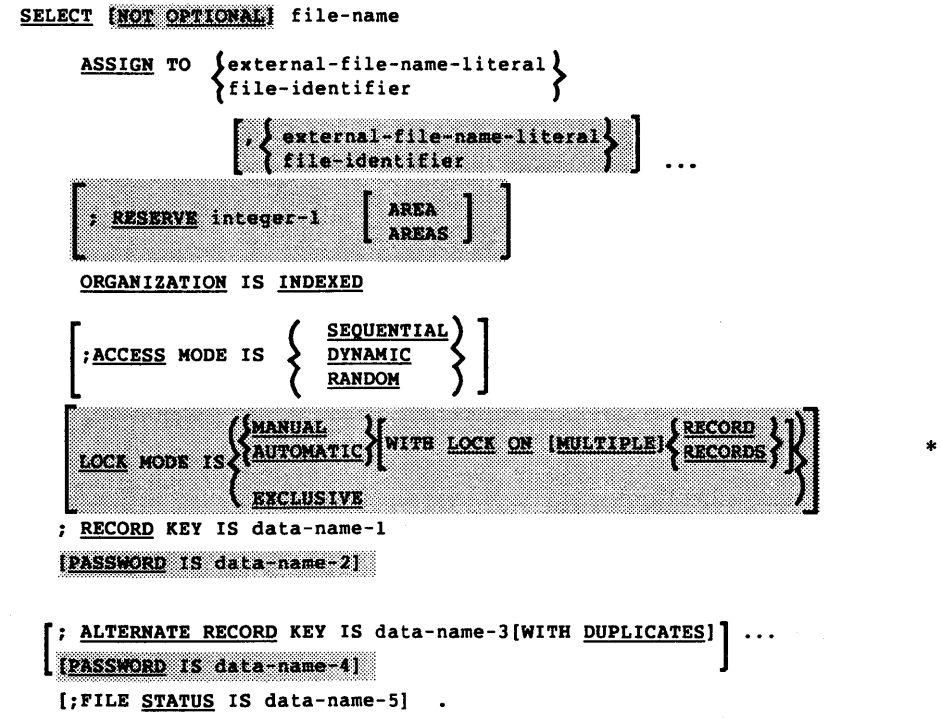
OPEN mode	No Directive	AUTOLOCK Directive	Override in SELECT Statement
INPUT	No lock	No lock	Yes, but only to EXCLUSIVE
I-O	Exclusive	Automatic lock on single record	Yes
OUTPUT	Exclusive	Exclusive	No

Note that:

- A file opened for OUTPUT causes the file to become exclusive, regardless of the specified lock mode.
- Explicitly or implicitly specifying automatic or manual record locking for a file causes the file to become shareable. A file opened for I-O acquires record locks, a file opened for INPUT never acquires record locks.
- The programmer can select the type of locking for individual files by accepting the default locking (see Table 7-1) or by including a LOCK MODE clause in the SELECT statement for a file (see **The SELECT Statement** later in this chapter).

In single user environments the multi-user syntax has no effect at run-time but programs can be developed for use in both single and multi-user environments.

General Format



Syntax Rules

1. The SELECT clause must be specified first in the file control entry. The clauses which follow the SELECT clause may appear in any order.

2. Each file described in the Data Division must be named once and only once as file-name in the FILE-CONTROL paragraph. Each file specified in the file control entry must have a file description entry in the Data Division.

3. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause is implied.

4. When the access mode is sequential, records in the file are accessed in the sequence dictated by the file organization. For indexed files this sequence is the order of ascending record key values within a given key of reference.

5. When the FILE STATUS clause is specified, a value will be moved by the operating system into the data item specified by data-name-5 after the execution of every statement that references that file either explicitly or implicitly. This value indicates the status of execution of the statement. (See **I-O Status** in this chapter.)

6. If the access mode is random, the value of the record key data item indicates the record to be accessed.

7. When the access mode is dynamic, records in the file may be accessed sequentially and/or randomly. (See general rules 4 and 6.)

8. The RECORD KEY clause specifies the record key that is the prime record key for the file. The values of the prime record key must be unique among records of the file. This prime record key provides an access path to records in an indexed file.

9. An ALTERNATE RECORD KEY clause specifies a record key that is an alternative record key for the file. This alternate record key provides an alternate access path to records in an indexed file.

10. The PASSWORD is data-name clause is for documentary purposes only. This syntax is valid only if the IBM compiler directive is specified at compile time.

11. The data description of data-name-1 and data-name-3 as well as relative locations within a record must be the same as that used when the file was created. The number of alternate keys for the file must also be the same as that used when the file was created.

12. The DUPLICATES phrase specifies that the value of the associated alternate record key may be duplicated within any of the records in the file. If the DUPLICATES phrase is not specified, the value of the associated alternate record key must not be duplicated among any of the records in the file.

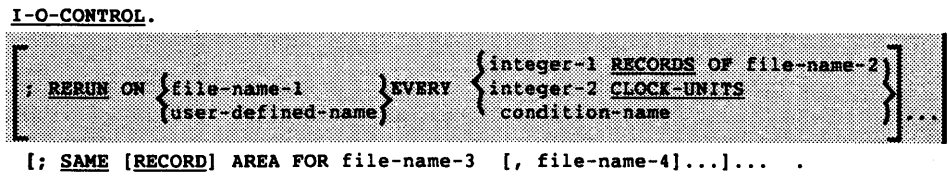
13. If file-identifier is not explicitly defined it will be implicitly defined.

THE I-O CONTROL PARAGRAPH

Function

The I-O-CONTROL paragraph specifies the points at which rerun is to be established and the memory area which is to be shared by different files.

General Format



Syntax Rules

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

2. file-name-1 must be a sequentially organized file.

3. When either the integer-1 RECORDS clause or the integer-2 CLOCK-UNITS clause is specified, user-defined-name must be given in the RERUN clause.

4. When multiple integer-1 RECORDS clauses are specified, no two of them may specify the same file-name-2.

5. Only one RERUN clause containing the CLOCK-UNITS clause may be specified.

3. The SAME RECORD AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. All of the files may be open at the same time. A logical record in the SAME RECORD AREA is considered as a logical record of each opened output file whose file-name appears in this SAME RECORD AREA clause and of the most recently read input file whose file-name appears in this SAME RECORD AREA clause. This is equivalent to an implicit redefinition of the area; that is, records are aligned on the leftmost character position.

DATA DIVISION IN THE INDEXED I-O MODULE

File Section

In a COBOL program the file description entry (FD) represents the highest level of organization in the File Section. The File Section header is followed by a file description entry consisting of a level indicator (FD), a file-name and a series of independent clauses. The FD clauses specify the size of the logical and physical records, the presence or absence of label records, the value of implementor-defined label items, and the names of the data records which comprise the file. The entry itself is terminated by a period.

Record Description Structure

A record description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name if required, followed by a series of independent clauses as required. A record description has a hierarchical structure and therefore the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description is defined in **Concepts of Levels** in Chapter 2 while the elements allowed in a record description are shown in **The Data Description - Complete Entry Skeleton** in Chapter 3.

The BLOCK CONTAINS Clause

FUNCTION

The BLOCK CONTAINS clause specifies the size of a physical record.

GENERAL FORMAT

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

GENERAL RULE

This clause is required for documentation purposes only.

GENERAL FORMAT

```
LABEL  { RECORD IS  }  { STANDARD }
       { RECORDS ARE}  { OMITTED  }
```

SYNTAX RULE

This clause is required in every file description entry, in ANSI standard COBOL, but optional in LEVEL II COBOL.

GENERAL RULE

This clause is used for documentation purposes only.

The RECORD CONTAINS Clause

FUNCTION

The RECORD CONTAINS clause specifies the size of data records.

GENERAL FORMAT

```
RECORD CONTAINS  [integer-1 TO]  integer-2 CHARACTERS
```

GENERAL RULE

The size of each data record is completely defined within the record description entry, therefore this clause is never required. The RECORD CONTAINS clause is specified for documentation purposes only.

For an output file, at the appropriate time the value of data-name-1 is made equal to the value of literal-1, or of a data-name-2, whichever has been specified.

3. A figurative constant may be substituted in the format above wherever a literal is specified.

PROCEDURE DIVISION IN THE INDEXED I-O MODULE

The CLOSE Statement

FUNCTION

The CLOSE statement terminates the processing of files.

GENERAL FORMAT

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

SYNTAX RULE

The files referenced in the CLOSE statement need not all have the same organization or access.

GENERAL RULES

1. A CLOSE statement may only be executed for a file in an open mode.

2. Indexed files are classified as belonging to the category of non-sequential single/multi-reel/unit. The results of executing each type of CLOSE for this category of file are summarized in the following table.

B. File Lock

This file cannot be opened again during this execution of this run unit.

3. The action taken if a file is in the open mode when a STOP RUN statement is executed is to close the file. The action taken for a file that has been opened in a called program and not closed in that program prior to the execution of a CANCEL statement for that program is to close the file.

4. If a CLOSE statement has been executed for a file, no other statement can be executed that references that file, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.

5. Following the successful execution of a CLOSE statement, the record area associated with file-name is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

6. If WITH LOCK is specified, the file cannot be reopened in the current execution of the run unit.

7. Following the successful execution of a CLOSE statement, all record or file locks held by the run unit on the closed file are released.

The DELETE Statement

FUNCTION

The DELETE statement logically removes a record from a mass storage file.

GENERAL FORMAT

```
DELETE file-name RECORD [; INVALID KEY imperative-statement]
```

SYNTAX RULES

1. The INVALID KEY phrase must not be specified for a DELETE statement which references a file which is in sequential access mode.

2. The INVALID KEY phrase must be specified for a DELETE statement which references a file which is not in sequential access mode and for which an applicable USE procedure is not specified.

GENERAL RULES

1. The associated file must be open in I-O mode at the time of the execution of this statement. (See **The OPEN Statement** later in this chapter.)

2. For files in the sequential access mode, the last input-output statement executed for file-name prior to the execution of the DELETE statement must have been a successfully executed READ statement. The operating system logically removes from the file the record that was accessed by that READ statement.

The OPEN Statement

FUNCTION

The OPEN statement initiates the processing of files. It also performs checking and/or writing of labels and other input-output operations.

GENERAL FORMAT

```
          ⎧ INPUT  file-name-1  [,file-name-2] ... ⎫
OPEN      ⎨ OUTPUT file-name-3  [,file-name-4] ... ⎬ ...
          ⎩ I-O    file-name-5  [,file-name-6] ... ⎭
```

SYNTAX RULE

The files referenced in the OPEN statement need not all have the same organization or access.

GENERAL RULES

1. The successful execution of the OPEN statement determines the availability of the file and results in the file being in an open mode.

2. The successful execution of the OPEN statement makes the associated record area available to the program.

3. Prior to the successful execution of an OPEN statement for a given file, no statement can be executed that references that file, either explicitly or implicitly.

4. An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. In Table 7-2, Permissible Statements, 'X' at an intersection indicates that the specified statement, used in the access mode given for that row, may be used with the indexed file organization and the open mode given at the top of the column.

8. The file description entry for file-name-1, file-name-2, file-name-5, or file-name-6 must be equivalent to that used when this file was created.

9. For files being opened with the INPUT or I-O phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. For indexed files, the prime record key is established as the key of reference and is used to determine the first record to be accessed. If no records exist in the file, the current record pointer is set such that the next executed Format 1 or Format 3 READ statement for the file will result in an AT END condition. If the file does not exist, INPUT will cause an error status.

10. The I-O phrase permits the opening of a file for both input and output operations. If the file does not exist, it will be created unless NOT OPTIONAL is specified in the SELECT statement. In sequential access mode it will then be used for input; any attempt to WRITE to it will cause an error.

11. When the I-O phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

    a. The labels are checked in accordance with the operating system specified conventions for input-output label checking.

    b. The new labels are written in accordance with the operating system specified conventions for input-output label writing.

12. Upon successful execution of an OPEN statement with the OUTPUT phrase specified, a file is created. At that time the associated file contains no data records. If a file of the same name exists it will be deleted. If write protected, an error status occurs.

13. The execution of the OPEN statement causes the value of the FILE STATUS data item to be updated (see I-O Status in this chapter).

The READ Statement

FUNCTION

For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a specified record from a mass storage file.

GENERAL FORMAT

Format 1

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

Format 2

```
READ file-name RECORD [INTO identifier]
     [WITH LOCK]    [; INVALID KEY imperative-statement
```

Format 3

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

Format 4

```
READ file-name RECORD [INTO identifier]
     [WITH [KEPT] LOCK]    [;INVALID KEY imperative-statement]
```

SYNTAX RULES

1. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the storage area which is the record area associated with file-name must not be the same storage area.

b. If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file with the established key of reference and then that record is made available.

3. The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. (See **I-O Status** in this chapter.)

4. Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged in that a record is available to the object program prior to the execution of any statement following the READ statement.

5. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.

6. If the INTO phrase is specified, the record being read is moved from the record area to the area specified by identifier according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

7. When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.

8. If, at the time of execution of a Format 1 or Format 3 READ statement, the position of current record pointer for that file is undefined, the execution of that READ statement is unsuccessful.

9. If, at the time of the execution of a Format 1 or Format 3 READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful. (See **I-O Status** in this chapter.)

15. If the KEY phrase is not specified in a Format 2 or Format 4 READ statement, the prime record key is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 or Format 3 READ statement for the file.

16. For an indexed file if the KEY phrase is specified in a Format 2 or Format 4 READ statement, data-name is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 or Format 3 READ statements for the file until a different key of reference is established for the file.

17. Execution of a Format 2 or Format 4 READ statement causes the value of the key of reference to be compared with the value contained in the corresponding data item of the stored records in the file, until the first record having an equal value is found. The current record pointer is positioned to this record which is then made available. If no record can be so identified, the INVALID KEY condition exists and execution of the READ statement is unsuccessful. (See **The INVALID KEY Condition** in this chapter.)

18. If the lock mode is AUTOMATIC with either single or multiple record locking, and the referenced file is opened I-O, the run unit acquires a record lock for the record retrieved by the successful execution of the READ statement.

19. If the lock mode is MANUAL with single record locking and the referenced file is opened I-O, the run unit acquires a record lock on the record only if the WITH LOCK phrase is specified. A simple READ statement does not acquire a record lock.

20. If the lock mode is MANUAL with multiple record locking and the referenced file is opened I-O, then the run unit acquires a lock on the record only if the WITH KEPT LOCK phrase is specified. A simple READ statement does not acquire a record lock.

21. Execution of a READ NEXT statement does not update the current record pointer if a locked record is found. A subsequent READ statement will attempt to read the same record.

NOTE:

To read past a locked record the current record pointer should be updated using the START statement.

4. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause, in which case the logical record is available to the program as a record of other files appearing in the same SAME RECORD AREA clause as the associated I-O file, as well as to the file associated with record-name.

5. The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of:

   MOVE identifier TO record-name

   followed by the execution of the same REWRITE statement without the FROM phrase. The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of the REWRITE statement.

6. The current record pointer is not affected by the execution of a REWRITE statement.

7. The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. (See **I-O Status.**)

8. For a file in the sequential access mode, the record to be replaced is specified by the value contained in the prime record key. When the REWRITE statement is executed the value contained in the prime record key data item of the record to be replaced must be equal to the value of the prime record key of the last record read from this file.

9. For a file in the random or dynamic access mode, the record to be replaced is specified by the prime record key data item.

10. The contents of alternative record key data items of the record being rewritten may differ from those in the record being replaced. The operating system utilizes the content of the record key data items during the execution of the REWRITE statement in such a way that subsequent access of the record may be made based upon any of those specified record keys.

The START Statement

FUNCTION

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

GENERAL FORMAT

```
START  file-name [ KEY  { IS EQUAL TO                } data-name ]
                        { IS =                       }
                        { IS GREATER THAN            }
                        { IS >                       }
                        { IS NOT LESS THAN           }
                        { IS NOT <                   }
      [;INVALID  KEY  imperative-statement]
```

NOTE:

The required relational characters '>', and '<' and '=' are not underlined to avoid confusion with other symbols such as '≥' (greater than or equal to).

SYNTAX RULES

1. file-name must be the name of an indexed file.

2. file-name must be the name of a file with sequential or dynamic access.

3. data-name may be qualified.

4. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.

5. If file-name is the name of an indexed file, and if the KEY phrase is specified, data-name may reference a data item specified as a record key associated with file-name, or it may reference any data item of category alpanumeric subordinate to the data-name of a data item specified as a record key associated with file-name whose leftmost character position corresponds to the leftmost character position of that record key data item.

b. If the KEY phrase is specified, and data-name is specified as a record key for file-name, that record key becomes the key of reference.

c. If the KEY phrase is specified, and data-name is not specified as a record key for file-name, the record key whose leftmost character position corresponds to the leftmost character position of the data item specified by data-name, becomes the key of reference.

8. If the execution of the START statement is not successful, the key of reference is undefined.

9. The START statement never acquires a record lock, nor does it detect a record lock.

The USE Statement

FUNCTION

The USE statement specifies procedures for input-output error handling that are in addition to the standard procedures provided by the input-output control system.

GENERAL FORMAT

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

SYNTAX RULES

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

2. The USE statement itself is never executed; it merely defines the conditions calling for the execution of the USE procedures.

3. The same file-name can appear in a different specific arrangement of the format. Appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE procedure.

4. The words ERROR and EXCEPTION are synonymous and may be used interchangeably.

5. The files implicitly referenced in a USE statement need not all have the same organization or access.

The WRITE Statement

FUNCTION

The WRITE statement releases a logical record for an output or input-output file.

GENERAL FORMAT

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

SYNTAX RULES

1. record-name and identifier must not reference the same storage area.

2. The record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

3. The INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated file.

GENERAL RULES

1. The associated file must be open in the OUTPUT or I-O mode at the time of the execution of this statement. (See **The OPEN Statement** in this chapter.)

2. The logical record released by the execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause or the execution of the WRITE statement is unsuccessful due to an INVALID KEY condition. The logical record is available to the program from the file associated with record-name and from other files referenced in the same SAME RECORD AREA clause as the associated output file.

11. The data item specified as the prime record key must be set by the program to the desired value prior to the execution of the WRITE statement.

12. If sequential access mode is specified for the file, records must be released to the operating system is ascending order of prime record key values.

13. If random or dynamic access mode is specified, records may be released to the operating system in any program-specified order.

14. When the ALTERNATE RECORD KEY clause is specified in the file control entry for an indexed file, the value of the alternate record key may be non-unique only if the DUPLICATES phrase is specified for that data item. In this case the operating system provides storage of records such that when records are accessed sequentially, the order of retrieval of those records is the order in which they are released to the operating system.

15. The INVALID KEY condition exists under the following circumstances:

    a. When sequential access mode is specified for a file opened in the output mode, and the value of the prime record key is not greater than the value of the prime record key of the previous record, or:

    b. When the file is opened in the output or I-O mode, and the value of the prime record key is equal to the value of a prime record key of a record already existing in the file, or:

    c. When the file is opened in the output or I-O mode, and the value of an alternate record key for which duplicates are not allowed equals the corresponding data item of a record already existing in the file, or:

    d. When an attempt is made to write beyond the externally defined boundaries of the file.

16. When the INVALID KEY condition is recognized the execution of the WRITE statement is unsuccessful, the contents of the record area are unaffected and the FILE STATUS data item, if any, associated with file-name of the associated file is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules stated under **The INVALID KEY Condition** (see also **I-O Status** in this chapter).

CHAPTER 8

SORT-MERGE

INTRODUCTION TO THE SORT-MERGE MODULE

The Sort-Merge module provides the capability to order one or more files of records, or to combine two or more identically ordered files of records, according to a set of user-specified keys contained within each record. Optionally, a user may apply some special processing to each of the individual records by input or output procedures. This special processing may be applied before and/or after the records are ordered by the SORT or after the records have been combined by the MERGE.

Relationship with Sequential I-O Module

The files specified in the USING and GIVING phrases of the SORT and MERGE statements must be described implicitly or explicity in the FILE-CONTROL paragraph as having sequential organization. No input-output statement may be executed for the file named in the sort-merge file description.

ENVIRONMENT DIVISION IN THE SORT-MERGE MODULE

Input-Output Section

THE FILE-CONTROL PARAGRAPH

Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

THE I-O-CONTROL PARAGRAPH

Function

The I-O-CONTROL paragraph specifies the memory area which is to be shared by different files.

General Format

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

Syntax Rules

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

2. In the SAME AREA clause, SORT and SORT-MERGE are equivalent.

3. If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file-names must represent a sort or merge file. Files that do not represent sort or merge files may also be named in the clause.

4. The three formats of the SAME clause (SAME RECORD AREA, SAME SORT AREA, SAME SORT-MERGE AREA) are considered separately in the following:

   More than one SAME clause may be included in a program, however:

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

   b. A file-name that represents a sort or merge file must not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.

c. Files other than sort or merge files do not share the same storage area with each other. If the user wishes these files to share the same storage area with each other, he must also include in the program a SAME AREA or SAME RECORD AREA clause naming these files.

d. During the execution of a SORT or MERGE statement that refers to a sort or merge file named in this clause, any non sort-merge files named in this clause must not be open.

DATA DIVISION IN THE SORT-MERGE MODULE

File Section

An SD file description gives information about the size and the names of the data records associated with the file to be sorted or merged. There are no label procedures which the user can control, and the rules for blocking and internal storage are peculiar to the SORT and MERGE statements.

The Sort-Merge File Description - Complete Entry Skeleton

FUNCTION

The sort-merge file description furnishes information concerning the physical structure, identification and record names of the file to be sorted or merged.

GENERAL FORMAT

```
SD file-name
    [; RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS ]

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

GENERAL RULES

1. The DATA RECORDS clause is treated as for documentation purposes only.

2. The presence of more than one data-name indicates that the file contains more than one type of data record. These records may be of differing sizes, different formats, etc. The order in which they are listed is not significant.

3. Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.

The RECORD CONTAINS Clause

FUNCTION

The RECORD CONTAINS clause specifies the size of data records.

GENERAL FORMAT

```
RECORD CONTAINS  [integer-1 TO]  integer-2 CHARACTERS
```

GENERAL RULES

1. The RECORD CONTAINS clause is treated as for documentation purposes only.

PROCEDURE DIVISION IN THE SORT-MERGE MODULE

The MERGE Statement

FUNCTION

The MERGE statement combines two or more identically sequenced files on a set of specified keys, and during the process makes records available, in merge order, to an output procedure or to an output file.

GENERAL FORMAT

```
MERGE file-name-1 ON  {ASCENDING }  KEY data-name-1 [, data-name-2]  ...
                      {DESCENDING}
                    [ ON {ASCENDING }  KEY data-name-3 [, data-name-4]...]...
                         {DESCENDING}
     [COLLATING SEQUENCE IS alphabet-name]
     USING file-name-2, file-name-3 [, file-name-4] ...
     { OUTPUT PROCEDURE IS section-name-1 [{THROUGH} section-name-2 ] }
     {                                    {THRU   }                 }
     { GIVING file-name-5                                           }
```

SYNTAX RULES

1. file-name-1 must be described in a sort-merge file description entry in the Data Divison.

2. section-name-1 represents the name of an output procedure.

GENERAL RULES

1. The MERGE statement will merge all records contained on file-name-2, file-name-3, and file-name-4. The files referenced in the MERGE statement must not be open at the time the MERGE statement is executed. These files are automatically opened and closed by the merge operation with all implicit functions performed, such as the execution of any associated USE procedures. The terminating function for all files is performed as if a CLOSE statement, without optional phrases, had been executed for each file.

2. The data-names following the word KEY are listed from left to right in the MERGE statement in order of decreasing significance without regard to how they are divided into KEY phrases. In the format data-name-1 is the major key, data-name-2 is the next most significant key, etc.

    a. When the ASCENDING phrase is specified, the merged sequence will be from the lowest value of the contents of the data items identified by the KEY data-names to the highest value, according to the rules for comparison of operands in a relation condition.

    b. When the DESCENDING phrase is specified, the merged sequence will be from the highest value of the contents of the data items identified by the KEY data-names to the lowest value, according to the rule for comparison of operands in a relation condition.

3. The collating sequence that applies to the comparison of the nonnumeric key data items specified is determined in the following order of precedence:

    a. First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in that MERGE statement.

    b. Second, the collating sequence established as the program collating sequence.

6. Segmentation, as defined in Chapter 9, can be applied to programs containing the MERGE statement. However, the following restrictions apply:

   a. If the MERGE statement appears in a section that is not in an independent segment, then any output procedure referenced by that MERGE statement must appear:

      - Totally within non-independent segments, or
      - Wholly contained in a single independent segment.

   b. If a MERGE statement appears in an independent segment, then any output procedure referenced by that MERGE statement must be contained:

      - Totally within non-independent segments, or
      - Wholly within the same independent segment as that MERGE statement.

7. If the GIVING phrase is specified, all the merged records in file-name-1 are automatically written on file-name-5 as the implied output procedure for this MERGE statement.

8. In the case of equal compare, according to the rules for comparison of operands in a relation condition, on the contents of the data items identified by all the KEY data-names between records from two or more input files (file-name-2, file-name-3, file-name-4, ...), the records are written on file-name-5 or returned to the output procedure, depending on the phrase specified, in the order that the associated input files are specified in the MERGE statement.

9. The results of the merge operation are predictable only when the records in the files referenced by file-name-2, file-name-3, ..., are ordered as described in the ASCENDING or DESCENDING KEY clause associated with the MERGE statement.

3. After the execution of the RELEASE statement, the logical record is no longer available in the record area unless the associated sort-merge file is named in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated sort-merge file, as well as to the file associated with record-name. When control passes from the input procedure, the file consists of all those records which were placed in it by the execution of RELEASE statements.

2. The execution of the RETURN statement causes the next record, in the order specified by the keys listed in the SORT or MERGE statement, to be made available for processing in the record areas associated with the sort or merge file.

3. If the INTO phrase is specified, the current record is moved from the input area to the area specified by identifier according to the rules for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if there is an AT END condition. Any subscripting or indexing associated with identifier is evaluated after the record has been returned and immediately before it is moved to the data item.

4. When the INTO phrase is used, the data is available in both the input record area and the data area associated with identifier.

5. If no next logical record exists for the file at the time of the execution of a RETURN statement, the AT END condition occurs. The contents of the record areas associated with the file when the AT END condition occurs are undefined. After the execution of the imperative-statement in the AT END phrase, no RETURN statement may be executed as part of the current output procedure.

3. file-name-2, file-name-3 and file-name-4 must be described in a file description entry, not in a sort-merge file description entry, in the Data Division. The actual size of the logical record(s) described for file-name-2, file-name-3 and file-name-4 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, it is the programmer's responsibility to describe the corresponding records in such a manner so as to cause equal amounts of character positions to be allocated for the corresponding records.

4. data-name-1, data-name-2, data-name-3, and data-name-4 are KEY data-names and are subject to the following rules:

   a. The data items identified by KEY data-names must be described in records associated with file-name-1.

   b. KEY data-names may be qualified.

   c. The data items identified by KEY data-names must not be variable length items.

   d. If file-name-1 has more than one record description, then the data items identified by KEY data-names need be described in only one of the record descriptions.

   e. None of the data items identified by KEY data-names can be described by an entry which either contains an OCCURS clause or is subordinate to an entry which contains an OCCURS clause.

5. The words THRU and THROUGH are equivalent.

6. SORT statements may appear anywhere except in the declaratives portion of the Procedure Division or in an input or output procedure associated with a SORT or MERGE statement.

7. No more than one file-name from a multiple file reel can appear in the SORT statement.

. The input procedure must consist of one or more sections that appear contiguously in a source program and do not form a part of any output procedure. In order to transfer records to the file referenced by file-name-1, the input procedure must include the execution of at least one RELEASE statement. Control must not be passed to the input procedure when a related SORT statement is being executed. The input procedure can include any procedures needed to select, create, or modify records. The restrictions on the procedural statements within the input procedure are as follows:

   a. The input procedure must not contain any SORT or MERGE statements.

   b. The input procedure must not contain any explicit transfers of control to points outside the input procedure; ALTER, GO TO, and PERFORM statements in the input procedure are not permitted to refer to procedure-names outside the input procedure. COBOL statements are allowed that will cause an implied transfer of control to declaratives.

   c. The remainder of the Procedure Division must not contain any transfers of control to points inside the input procedure; ALTER, GO TO and PERFORM statements in the remainder of the Procedure Division must not refer to procedure-names within the input procedure.

5. If an input procedure is specified, control is passed to the input procedure before file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the input procedure and when control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted.

6. The output procedure must consist of one or more sections that appear contiguously in a source program and do not form part of any input procedure. In order to make sorted records available for processing, the output procedure must include the execution of at least one RETURN statement. Control must not be passed to the output procedure except when a related SORT statement is being executed. The output procedure may consist of any procedures needed to select, modify or copy the records that are being returned, one at a time in sorted order, from the sort file. The restrictions on the procedural statements within the output procedure are as follows:

   a. The output procedure must not contain any SORT or MERGE statements.

9. If the USING phrase is specified, all the records in file-name-2 and file-name-3 are transferred automatically to file-name-1. At the times of execution of the SORT statement, file-name-2 and file-name-3 must not be open. The SORT statement automatically initiates the processing of, makes available the logical records for, and terminates the processing of file-name-2 and file-name-3. These implicit functions are performed such that any associated USE Procedures are executed. The terminating function for all files is performed as if a CLOSE statement, without optional phrases, had been executed for each file. The SORT statement also automatically performs the implicit functions of moving the records from the file area of file-name-2 and file-name-3 to the file area for file-name-1 and the release of records to the initial phase of the sort operation.

10. If the GIVING phrase is specified, all the sorted records in file-name-1 are automatically written on file-name-4 as the implied output procedure for this SORT statement. At the time of execution of the SORT statement file-name-4 must not be open. The SORT statement automatically initiates the processing of, releases the logical records to, and terminates the processing of file-name-4. These implicit functions are performed such that any associated USE procedures are executed. The terminating function is performed as if a CLOSE statement, without optional phrases, had been executed for the file. The SORT statement also automatically performs the implicit functions of the return of the sorted records from the final phase of the sort operation and the moving of the records from the file area for file-name-1 to the file area for file-name-4.

CHAPTER 9

SEGMENTATION

INTRODUCTION TO THE SEGMENTATION MODULE

The Segmentation module provides a capability to specify object program overlay requirements.

Segmentation provides a facility for specifying permanent and independent segments. All segments specified as permanent segments must be contiguous in the source program. Segmentation also allows the intermixing of sections with different segment-numbers and allows the fixed portion of the source program to contain segments that may be overlaid.

GENERAL DESCRIPTION OF SEGMENTATION

COBOL segmentation is a facility that provides a means by which the user may communicate with the compiler to specify object program overlay requirements.

COBOL segmentation deals only with segmentation of procedures. As such, only the Procedure Division is considered in determining segmentation requirements for an object program.

INDEPENDENT SEGMENTS

An independent segment is defined as part of the object program which can overlay, and can be overlaid by either a fixed overlayable segment or another independent segment. An independent segment is in its initial state whenever control is transferred (either implicitly or explicitly) to that segment for the first time during the execution of a program. On subsequent transfers of control to the segment, an independent segment is also in its initial state when:

1. Control is transferred to that segment as a result of the implicit transfer of control between consecutive statements from a segment with a different segment-number.

2. Control is transferred to that segment as the result of the implicit transfer of control between a SORT or MERGE statement, in a segment with a different segment-number, and an associated input or output procedure in that independent segment.

3. Control is transferred explicitly to that segment from a segment with a different segment-number (with the exception noted in paragraph 2 below).

On subsequent transfer of control to the segment, an independent segment is in its last-used state when:

1. Control is transferred implicitly to that segment from a segment with a different segment-number (except as noted in paragraph 1).

2. Control is transferred explicitly to that segment as the result of the execution of an EXIT PROGRAM statement.

Segmentation Classification

Sections which are to be segmented are classified, using a system of segment-numbers and the following criteria:

1. Logic Requirements - Sections which must be available for reference at all times, or which are referred to very frequently, are normally classified as belonging either to one of the overlayable fixed segments or to one of the permanent segments; sections which are used less frequently are normally classified as belonging to one of the independent segments, depending on logic requirements.

3. Sections in the declaratives must contain segment-numbers less than 50.

GENERAL RULES

1. All sections which have the same segment-number constitute a program segment. All sections which have the same segment-number need not be physically contiguous in the source program.

2. Segments with segment-number 0 through 49 belong to the fixed portion of the object program.

3. Segments with segment-number 50 through 99 are independent segments.

The SEGMENT LIMIT Clause

GENERAL FORMAT

The SEGMENT-LIMIT clause appears in the OBJECT-COMPUTER paragraph and has the following format:

```
[, SEGMENT-LIMIT IS segment-number]
```

SYNTAX RULES

segment-number must be an integer ranging in value from 1 through 49.

GENERAL RULES

1. The SEGMENT-LIMIT clause is treated as for documentation purposes only.

A PERFORM statement that appears in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

a. Sections and/or paragraphs wholly contained in one or more non-independent segments.

b. Sections and/or paragraphs wholly contained in the same independent segment as that PERFORM statement.

The MERGE Statement

If the MERGE statement appears in a section that is not in an independent segment, then any output procedure referenced by that MERGE statement must appear:

a. Totally within non-independent segments, or

b. Wholly contained in a single independent segment.

If a MERGE statement appears in an independent segment, then any output procedure referenced by that MERGE statement must be contained:

a. Totally within non-independent segments, or

b. Wholly within the same independent segment as that MERGE statement.

The SORT Statement

If a SORT statement appears in a section that is not an independent segment, then any input procedures or output procedures referenced by that SORT statement must appear:

a. Totally within non-independent segments, or

b. Wholly contained in a single independent segment.

CHAPTER 10

LIBRARY

INTRODUCTION TO THE LIBRARY MODULE

The Library module provides a capability for specifying text that is to be copied from a source user-library file. This is usually created using any suitable source text editor.

LEVEL II COBOL libraries consist of disk files that contain source to be made available to the compiler. The effect of the interpretation of the COPY statement is to insert text into the source program, where it will be treated by the compiler as part of the source program. All occurrences of a given literal, identifier, word or group of words in the library text can be replaced with alternate text during the copy process. The library module also provides for the availability of more than one COBOL library at compile time.

7. A COPY statement may occur in the source program anywhere a character-string or a separator may occur except that a COPY statement must not occur within a COPY statement.

8. text-name defines a unique external file name which conforms to the rules for user defined words (note lower case is translated to upper case). external-file-name-literal is an alphanumeric literal enclosed in quotes that conforms to the operating system rules for file names.

9. library-name-literal is an alphanumeric literal enclosed in quotes that conforms either to the operating system rules for file names or to the operating system rules for device identifiers. The exact format for file names and device identifiers depends on the operating system. Refer to the **LEVEL II COBOL Operating Guide** for details.

GENERAL RULES

1. The compilation of a source program containing COPY statement is logically equivalent to processing all COPY statements prior to the processing of the resulting source program.

2. The effect of processing a COPY statement is that the library text associated with text-name is copied into the source program, logically replacing the entire COPY statement, beginning with the reserved word COPY and ending with the punctuation character period, inclusive.

3. If the REPLACING phrase is not specified, the library text is copied unchanged.

   If the REPLACING phrase is specified, the library text is copied and each properly matched occurrence of pseudo-text-1, identifier-1, word-1, and literal-1 in the library text is replaced by the corresponding pseudo-text-2, identifier-2, word-2, or literal-2.

4. For purposes of matching, identifier-1, word-1, and literal-1 are treated as pseudo-text containing only identifier-1, word-1, or literal-1, respectively.

6. A comment line occurring in the library text and pseudo-text-1 is interpreted, for purposes of matching, as a single space. Comment lines appearing in pseudo-text-2 and library text are copied into the source program unchanged.

7. Debugging lines are permitted within library text and pseudo-text-2. Debugging lines are not permitted within pseudo-text-1; text-words within a debugging line participate in the matching rules as if the 'D' did not appear in the indicator area. If a COPY statement is specified on a debugging line, then the text that is the result of the processing of the COPY statement will appear as though it were specified on debugging lines with the following exception: comment lines in library text will appear as comment lines in the resultant source program.

8. The text produced as a result of the complete processing of a COPY statement must not contain a COPY statement.

9. The syntatic correctness of the library text cannot be independently determined. The syntatic correctness of the entire COBOL source program cannot be determined until all COPY statements have been completely processed.

10. Library text must conform to the rules for COBOL reference format.

11. For purposes of compilation, text-words after replacement are placed in the source program according to the rules for reference format as described in Chapter 1.

12. If the unit identifier is not explicitly specified, the default drive will be used. The default is operating system dependent and is described in the **LEVEL II COBOL Operating Guide.**

CHAPTER 11

DEBUG AND INTERACTIVE DEBUGGING

INTRODUCTION

Standard ANSI COBOL debugging provides a means by which the user can describe the conditions under which procedures are to be monitored during the execution of the object program.

The optional ANIMATOR debugging product is also available, and brings a program to life on the screen *animating* it by displaying the source code during run time with the cursor moving from COBOL source statement to statement. ANIMATOR is a fully interactive symbolic debugging tool that complies with the published GSA certification standard enabling the setting of breakpoints, examination and alteration of data and the changing of the flow of control. It is supplied with its own manual.

This Chapter describes the standard ANSI '74 COBOL DEBUG module.

STANDARD ANSI COBOL DEBUG

The decisions of what to monitor and what information to display are explicitly in the domain of the user. The COBOL Debug facility simply provides a convenient access to pertinent information.

The features of the language that support the COBOL Debug module are:

- A compile time switch -- WITH DEBUGGING MODE.
- An object time switch.
- A USE FOR DEBUGGING statement.
- A special register -- DEBUG-ITEM.
- Debugging lines.

Environment Division in COBOL DEBUG

THE WITH DEBUGGING MODE CLAUSE

Function

The WITH DEBUGGING MODE clause indicates that all debugging sections and all debugging lines are to be compiled. If this clause is not specified, all debugging lines and sections are compiled as if they were comment lines.

General Format

```
SOURCE-COMPUTER.  computer-name    [WITH DEBUGGING MODE].
```

General Rules

1. If the WITH DEBUGGING MODE clause is specified in the SOURCE-COMPUTER paragraph of the Configuration Section of a program, all USE FOR DEBUGGING statements and all debugging lines are compiled.

2. If the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph of the Configuration Section of a program, any USE FOR DEBUGGING statements and all associated debugging sections, and any debugging lines are compiled as if they were comment statements.

4. Except for the USE FOR DEBUGGING statement itself, statements appearing within a given debugging section may reference procedure-names defined within a different USE procedure only with a PERFORM statement.

5. Procedure-names defined within debugging sections must not appear within USE FOR DEBUGGING statements.

6. Any given identifier, cd-name, file-name, or procedure-name may appear in only one USE FOR DEBUGGING statement and may appear only once in that statement.

7. The ALL PROCEDURES phrase can appear only once in a program.

8. When the ALL PROCEDURES phrase is specified, procedure-name-1, procedure-name-2, ... must not be specified in any USE FOR DEBUGGING statement.

9. If the data description entry of the data item referenced by identifier-1, identifier-2, ..., contains an OCCURS clause or is subordinate to a data description entry that contains an OCCURS clause, identifier-1, identifier-2, ..., must be specified without the subscripting or indexing normally required.

10. References to the special register DEBUG-ITEM are restricted to references from within a debugging section.

General Rules

1. In the following general rules all references to cd-name-1, identifier-1, procedure-name-1, and file-name-1 apply equally to cd-name-2, identifier-2, procedure-name-2 and file-name-2 respectively.

2. Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.

3. When file-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:

   a. After the execution of any OPEN or CLOSE statement that references file-name-1, and

7. When identifier-1 is specified without the ALL REFERENCES OF phrase, that debugging section is executed at each of the following times:

   a. In the case of a WRITE or REWRITE statement that explicitly references identifier-2, immediately before the execution of that WRITE or REWRITE statement and after the execution of any implicit move resulting from the presence of the FROM phrase.

   b. In the case of a PERFORM statement in which a VARYING, AFTER or UNTIL phrase references identifier-1, immediately after each initialization, modification or evaluation of the contents of the data item referenced by identifier-1.

   c. Immediately after the execution of any other COBOL statement that explicitly references and causes the contents of the data item referenced by identifier-1 to be changed.

   If identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

8. The associated debugging section is not executed for a specific operand more than once as a result of the execution of a single statement, regardless of the number of times that operand is explicitly specified. In the case of a PERFORM statement which caused iterative execution of a referenced procedure, the associated debugging section is executed once for each iteration.

   Within an imperative statement, each individual occurrence of an imperative verb identifies a separate statement for the purpose of debugging.

9. When cd-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:

   a. After the execution of any ENABLE, DISABLE, and SEND statement that references cd-name-1,

   b. After the execution of a RECEIVE statement referencing cd-name-1 that does not result in the execution of the NO DATA imperative-statement, and

   c. After the execution of an ACCEPT MESSAGE COUNT statement that references cd-name-1.

All qualifiers of the name are separated in DEBUG-NAME by the word IN or OF.

Subscripts/indices, if any, are not entered into DEBUG-NAME.

15. If the reference to a data item that causes the debugging section to be executed is subscripted or indexed, the occurrence number of each level is entered in DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3 respectively as necessary.

16. DEBUG-CONTENTS is a data item that is large enough to contain the data required by the following general rules.

17. If the first execution of the first nondeclarative procedure in the program causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the first statement of that procedure.

    b. DEBUG-NAME contains the name of that procedure.

    c. DEBUG-CONTENTS contains 'START PROGRAM'.

18. If a reference to procedure-name-1 in an ALTER statement causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the ALTER statement that references procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains the applicable procedure-name associated with the TO phrase of the ALTER statement.

19. If the transfer of control associated with the execution of a GO TO statement causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the GO TO statement whose execution transfers control to procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

23. If an implicit transfer of control from the previous sequential paragraph to procedure-name-1 causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the previous statement.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains 'FALL THROUGH'.

24. If references to file-name-1, cd-name-1 causes the debugging section to be executed, then:

    a. DEBUG-LINE identifies the source statement that references file-name-1, cd-name-1.

    b. DEBUG-NAME contains the name of file-name-1, cd-name-1.

    c. For READ, DEBUG-CONTENTS contains the entire record read.

    d. For all other references to file-name-1, DEBUG-CONTENTS contains spaces.

    e. For any reference cd-name-1, DEBUG-CONTENTS contains the contents of the area associated with the cd-name.

25. If a reference to identifier-1 causes the debugging section to be executed, then:

    a. DEBUG-LINE identifies the source statement that references identifier-1,

    b. DEBUG-NAME contains the name of identifier-1, and

    c. DEBUG-CONTENTS contains the contents of the data item referenced by identifier-2 at the time that control passes to the debugging section (see General Rules 6 and 7).

CHAPTER 12

INTERPROGRAM COMMUNICATION

INTRODUCTION TO THE INTER-PROGRAM COMMUNICATION MODULE

The Inter-Program Communication module provides a facility by which a program can communicate with one or more programs. This provides a programmer with a modular programming capability. Each module when CALLed is loaded dynamically by the Run-Time System. Communication is provided by:

- The ability to transfer control from one program to another within a run unit
- The ability for both programs to have access to the same data items.

DATA DIVISION IN THE INTER-PROGRAM COMMUNICATION MODULE

Linkage Section

The Linkage Section in a program is meaningful if and only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

The Linkage Section is used for describing data that is available through the calling program but is to be referred to in both the calling and the called program. No space is allocated in the program for data items referenced by data-names in the Linkage Section of that program. Procedure Division references to these data items are resolved at object time by equating the reference in the called program to the location used in the calling program. In the case of index-names, no such correspondence is established. Index-names in the called and calling program always refer to separate indices.

Linkage Records

Data elements in the Linkage Section which bear a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of record descriptions. Any clause which is used in an input or output record description can be used in a Linkage Section.

Initial Values

The VALUE clause must not be specified in the Linkage Section except in condition-name entries (level 88).

PROCEDURE DIVISION IN THE INTER-PROGRAM COMMUNICATION MODULE

The Procedure Division Header

The Procedure Division is identified by and must begin with the following header:

```
PROCEDURE DIVISION   [USING data-name-1   [,  data-name-2]  ...]    .
```

The USING phrase is present if and only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

Each of the operands in the USING phrase of the Procedure Division header must be defined as a data item in the Linkage Section of the program in which this header occurs, and it must have a 01 or 77 level-number.

Within a called program, Linkage Section data items are processed according to their data descriptions given in the called program.

The CALL Statement

FUNCTION

The CALL statement causes control to be transferred from one object program to another, within the run unit.

GENERAL FORMAT

Format 1

```
CALL { identifier-1 } [USING  data-name-1    [, data-name-2]   ...]
     { literal-1    }

          [; ON OVERFLOW  imperative-statement]
```

Format 2

```
CALL { literal-2    } [USING  data-name-3    [, data-name-4]   ...]
     { identifier-2 }
```

SYNTAX RULES

1. literal-1must be a nonnumeric literal which identifies a file containing the subprogram to be CALLed.

2. identifier-1 must be defined as an alphanumeric data item whose usage is DISPLAY

3. The USING phrase is included in the CALL statement only if there is a USING phrase in the Procedure Division header of the called program and the number of operands in each USING phrase must be identical.

5. If during the execution of a CALL statement, it is determined that the available portion of run-time memory is incapable of accommodating the program specified in the CALL statement, the next sequential instruction is executed. If ON OVERFLOW has been specified, the associated imperative statement is executed before the next instruction is executed.

6. Called programs may contain CALL statements. However, a called program must not contain a call statement that directly or indirectly calls the calling program.

7. The data-names, specified by the USING phrase of the CALL statement, indicate those data items available to a calling program that may be referred to in the called program. The order of appearance of the data-names in the USING phrase of the CALL statement and the USING phrase in the Procedure Division header is critical. Corresponding data-names refer to a single set of data which is available to the called and calling program. The correspondence is positional, not by name. In the case of index-names, no such correspondence is established. Index-names in the called and calling program always refer to separate indices.

8. The CALL statement may appear anywhere within a segmented program. Therefore, when a CALL statement appears in a section with a segment-number greater than or equal to 50, that segment is in its last used state when the EXIT PROGRAM statement returns control to the calling program.

4. A called program is cancelled either by being referred to as the operand of a CANCEL statement or by the termination of the run unit of which the program is a member.

5. No action is taken when a CANCEL statement is executed naming a program that has not been called in this run unit or has been called and is at present cancelled. Control passes to the next statement.

4. When the USING phrase is present, the object program operates as if data-name-1 of the point of entry in the called program and data-name-1 in the USING phrase of the ENTRY statement refer to a single set of data that is equally available to both the called and calling programs. Their descriptions must define an equal number of character positions; however they need not be the same name. Similarly, there is an equivalent relationship between data-name-2 in both USING phrases.

5. A data-name must not appear more than once in the USING phrase in the ENTRY statement. However, a given data-name may appear more than once in the same USING phrase of a CALL statement.

6. If the USING phrase is specified, the INITIAL clause must not be present in any CD entry. (See **The Communication Description - Complete Entry Skeleton** in Chapter 13).

7. The number of operands in the ENTRY USING statement and the CALL USING statement may be different. If they are different, then the operands in the two statements are matched from left to right until the shorter list of operands is exhausted.

   If the remaining unmatched operands are in the CALL statement they are ignored. If the remaining unmatched operands are in the ENTRY statement they are unavilable to the program, and so if they are referenced at run-time, a run-time error will result.

   Any mis-match of operands will be detected only at run-time. The FLAG compiler directive does not cause this LEVEL II COBOL extension to be flagged.

The GOBACK Statement

FUNCTION

The GOBACK statement causes execution to return from a subprogram to the calling program, or to the operating system if the program has not been called.

GENERAL FORMAT

GOBACK.

SYNTAX RULE

The GOBACK statement must appear either as the only statement in a sentence, or as the last in a series of imperative statements in a sentence.

GENERAL RULE

The GOBACK statement is valid only if the IBM compiler directive is on (see the LEVEL II COBOL Operating Guide for details). When this directive is on, GOBACK is a reserved word.

CHAPTER 13

COMMUNICATION

INTRODUCTION TO THE COMMUNICATION MODULE

Function

The communication module provides the ability to access, process, and create messages or portions thereof. It provides the ability to communicate through a Message Control System (MCS) with local and remote communication devices.

DATA DIVISION IN THE COMMUNICATION MODULE

Communication Section

In a COBOL program the communication description entries (CD) represent the highest level of organization in the Communication Section. The Communication Section header is followed by a communication description entry consisting of a level indicator (CD), a data-name and a series of independent clauses. These clauses indicate the queues and sub-queues, the message date and time, the source, the text length, the status and end keys, and message count of input. These clauses specify the destination count, the text length, the status and error keys, and destinations for output. The entry itself is terminated by a period. These record areas may be implicitly redefined by user-specified record description entries following the various communication description clauses.

SYNTAX RULES

Format 1:

1. A CD must appear only in the Communication Section.

2. Within a single program, the INITIAL clause may be specified in only one CD. The INITIAL clause must not be used in a program that specifies the USING phrase of the Procedure Division Header. (See **The Procedure Division Header.**)

3. Except for the INITIAL clause, the optional clauses may be written in any order.

4. If neither option in the format is specified, a level 01 data description entry must follow the CD description entry. Either option may be followed by a level 01 data description entry.

5. For each input CD, a record area of 87 contiguous standard data format characters is allocated. This record area is defined to the MCS as follows:

   a. The SYMBOLIC QUEUE clause defines data-name-1 as the name of an elementary alphanumeric data item of 12 characters occupying positions 1-12 in the record.

   b. The SYMBOLIC SUB-QUEUE-1 clause defines data-name-2 as the name of an elementary alphanumeric data item of 12 characters occupying positions 13-24 in the record.

   c. The SYMBOLIC SUB-QUEUE-2 clause defines data-name-3 as the name of an elementary alphanumeric data item of 12 characters occupying positions 25-36 in the record.

   d. The SYMBOLIC SUB-QUEUE-3 clause defines data-name-4 as the name of an elementary alphanumeric data item of 12 characters occupying positions 37-48 in the record.

   e. The MESSAGE DATE clause defines data-name-5 as the name of a data item whose implicit description is that of an integer of 6 digits without an operational sign occupying character positions 49-54 in the record.

Use of either option results in a record whose implicit description is equivalent to the following:

```
                    IMPLICIT DESCRIPTION                    COMMENT

01   data-name-0.

     02   data-name-1    PICTURE X(12).      SYMBOLIC QUEUE
     02   data-name-2    PICTURE X(12).      SYMBOLIC SUB-QUEUE-1
     02   data-name-3    PICTURE X(12).      SYMBOLIC SUB-QUEUE-2
     02   data-name-4    PICTURE X(12).      SYMBOLIC SUB-QUEUE-3
     02   data-name-5    PICTURE 9(06).      MESSAGE DATE
     02   data-name-6    PICTURE 9(08).      MESSAGE TIME
     02   data-name-7    PICTURE X(12).      SYMBOLIC SOURCE
     02   data-name-8    PICTURE 9(04).      TEXT LENGTH
     02   data-name-9    PICTURE X.          END KEY
     02   data-name-10   PICTURE XX.         STATUS KEY
     02   data-name-11   PICTURE 9(06).      MESSAGE COUNT
```

NOTE:

In the above, the information under 'COMMENT' is for clarification and is not part of the description.

6. Record description entries following an input CD implicitly redefine this record and must describe a record of exactly 87 characters. Multiple redefinitions of this record are permitted; however, only the first redefinition may contain VALUE clauses. However, the MCS will always reference the record according to the data descriptions defined in syntax rule 5.

7. data-name-1, data-name-2, ..., data-name-11 must be unique within the CD. Within this series, any data-name may be replaced by the reserved word FILLER.

Format 2:

8. A CD must appear only in the Communication Section.

9. If none of the optional clauses of the CD is specified, a level 01 data description entry must follow the CD description entry.

NOTE:

In the above, the information under 'COMMENT' is for clarification and is not part of the description.

11. Record descriptions following an output CD implictly redefine this record. Multiple redefinitions of this record are permitted; however, only the first redefinition may contain VALUE clauses. Note that the MCS will always reference the record according to the data descriptions defined in syntax rule 10.

12. data-name-1, data-name-2, ..., data-name-5 must be unique within a CD.

13. If the DESTINATION TABLE OCCURS clause is not specified, one ERROR KEY and one SYMBOLIC DESTINATION area is assumed. In this case, neither subscripting nor indexing is permitted when referencing these data items.

14. If the DESTINATION TABLE OCCURS clause is specified, data-name-4 and data-name-5 may only be referred to by subscripting or indexing.

15. There is no restriction on the value of the data item referenced by data-name-1 and integer-2.

GENERAL RULES

Format 1:

1. The input CD information constitutes the communication between the MCS and the program as information about the message being handled. This information does not come from the terminal as part of the message.

2. The contents of the data items referenced by data-name-2, data-name-3, and data-name-4, when not being used must contain spaces.

3. The data items referenced by data-name-1, data-name-2, data-name-3, and data-name-4 contain symbolic names designating queues, sub-queues, ... respectively. All symbolic names must follow the rules for the formation of system-names, and must have been previously defined to the MCS.

8. The contents of data-name-6 have the format HHMMSSTT (hours, minutes, seconds, hundredths of a second) and its contents represent the time at which the MCS recognizes that the message is complete.

   The contents of the data item referenced by data-name-6 are only updated by the MCS as part of the execution of the RECEIVE statement.

9. During the execution of a RECEIVE statement, the MCS provides, in the data item referenced by data-name-7, the symbolic name of the communications terminal that is the source of the message being transferred. This symbolic move must follow the rules for the formation of system names. However, if the symbolic name of the communication terminal is not known to the MCS, the contents of the data item referenced by data-name-7 will contain spaces.

10. The MCS indicates via the contents of the data item referenced by data-name-8 the number of character positions filled as a result of the execution of the RECEIVE statement. (See **The RECEIVE Statement** later in this Chapter.)

11. The contents of the data item referenced by data-name-9 are set only by the MCS as part of the execution of a RECEIVE statement according to the following rules:

    a. When the RECEIVE MESSAGE phrase is specified, then data-name-9 is set to one of the following:

       - If an end of group has been detected, the contents of the data item referenced by data-name-9 are set to 3;
       - If an end of message has been detected, the contents of the data item referenced by data-name-9 are set to 2;
       - If less than a message has been detected, the contents of the data item referenced by data-name-9 are set to 0.

    b. When the RECEIVE SEGMENT phrase is specified, data-name-9 is set to one of the following:

       - If an end of group has been detected, the contents of the data item referenced by data-name-9 are set to 3;
       - If an end of message has been detected, the contents of the data item referenced by data-name-9 are set to 2;

If during the execution of a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement the value of the data item referenced by data-name-1 is outside the range of 1 through integer-2, an error condition is indicated and the execution of the SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement is terminated.

16. It is the responsibility of the user to insure that the value of the data item referenced by data-name-1 is valid at the time of execution of the SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement.

17. As part of the execution of a SEND statement, the MCS will interpret the contents of the data item referenced by data-name-2 to be the user's indication of the number of leftmost character positions of the data item referenced by the associated SEND identififer from which data is to be transferred. (See **The SEND Statement** later in this Chapter).

18. Each occurrence of the data item referenced by data-name-5 contains a symbolic destination previously known to the MCS. These symbolic destination names must follow the rules for the formation of system-names.

19. The contents of the data item referenced by data-name-3 indicate the status condition of the previously executed SEND, ENABLE OUTPUT or DISABLE OUTPUT statement.

    The actual association between the contents of the data item referenced by data-name-3 and the status condition itself is defined in Table 13-1.

20. If, during the execution of a SEND, an ENABLE OUTPUT, or a DISABLE OUTPUT statement, the MCS determines that any specified destination is unknown, the contents of the data item referenced by data-name-3 and all occurrences of the data items referenced by data-name-4 are updated.

    The contents of the data item referenced by data-name-4 when equal to 1 indicate that the associated value in the area referenced by data-name-5 has not been previously defined to the MCS. Otherwise, the contents of the data item referenced by data-name-4 are set to zero.

PROCEDURE DIVISION IN THE COMMUNICATION MODULE

The ACCEPT MESSAGE COUNT Statement

FUNCTION

The ACCEPT MESSAGE COUNT statement causes the number of messages in a queue to be made available.

GENERAL FORMAT

```
ACCEPT cd-name MESSAGE COUNT
```

SYNTAX RULE

cd-name must reference an input CD.

GENERAL RULES

1. The ACCEPT MESSAGE COUNT statement causes the MESSAGE COUNT field specified for cd-name to be updated to indicate the number of messages that exist in a queue, sub-queue-1, ...

2. Upon execution of the ACCEPT MESSAGE COUNT statement, the contents of the area specified by a communication description entry must contain at least the name of the symbolic queue to be tested. Testing the condition causes the contents of the data items referenced by data-name-10 (STATUS KEY) and data-name-1 (MESSAGE COUNT) of the area associated with the communication entry to be appropriately updated. (See **The Communication Description - Complete Entry Skeleton.**)

3. When the INPUT phrase without the optional word TERMINAL is specified, the logical paths for all of the sources associated with the queues and sub-queues specified by the contents of data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name are deactivated.

4. When the OUTPUT phrase is specified, the logical path for destination, or the logical paths for all destinations, specified by the contents of the data item referenced by data-name-5 (SYMBOLIC DESTINATION) of the area referenced by cd-name are deactivated.

5. literal-1 or the contents of the data-name referenced by identifier-1 will be matched with a password built into the system. The DISABLE statement will be honored only if literal-1 or the contents of the data item referenced by identifier-1 match the system password. When literal-1 or the contents of the data item referenced by identifier-1 do not match the system password, the value of the STATUS KEY item in the area referenced by cd-name is updated.

   The MCS must be capable of handling a password of from one to ten characters inclusive.

6. The MCS will insure that the execution of a DISABLE statement will cause the logical disconnection at the earliest time the source or destination is inactive. The execution of the DISABLE statement will never cause the remaining portion of the message to be terminated during transmission to or from a terminal.

3. When the INPUT phrase without the optional word TERMINAL is specified, the logical paths for all of the sources associated with the queue and sub-queues specified by the contents of data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name are activated.

4. When the OUTPUT phrase is specified, the logical path for destination, or the logical paths for all destinations, specified by the contents of the data item referenced by data-name-5 (SYMBOLIC DESTINATION) of the area referenced by cd-name are activated.

5. literal-1 or the contents of the data item referenced by identifier-1 will be matched with a password built into the system. The ENABLE statement will be honored only if literal-1 or the contents of the data item referenced by identifier-1 match the system password. When literal-1 or the contents of the data item referenced by identifier-1 do not match the system password, the value of the STATUS KEY item in the area referenced by cd-name is updated.

   The MCS must be capable of handling a password of from one to ten characters inclusive.

3. When during the execution of a RECEIVE statement, the MCS makes data available in the data item referenced by identifier-1, control is transferred to the next executable statement, whether or not the NO DATA phrase is specified.

4. When, during the execution of a RECEIVE statement, the MCS does not make data available in the data item referenced by identifier-1:

   a. If the NO DATA phrase is specified, the RECEIVE operation is terminated with the indication that action is complete (see general rule 5), and the imperative statement in the NO DATA phrase is executed.

   b. If the NO DATA phrase is not specified, execution of the object program is suspended until data is made available in the data item referenced by identifier-1.

   c. If one or more queues or sub-queues is unknown to the MCS, control passes to the next executable statement, whether or not the NO DATA phrase is specified. (See Table 13-1 for Status.)

5. The data items identified by the input CD are appropriately updated by the Message Control System at each execution of a RECEIVE statement.

6. A single execution of a RECEIVE statement never returns to the data item referenced by identifier-1 more than a single message (when the MESSAGE phrase is used) or a single segment (when the SEGMENT phrase is used). However, the MCS does not pass any portion of a message to the object program until the entire message is available in the input queue, even if the SEGMENT phrase of the RECEIVE statement is specified.

7. When the MESSAGE phrase is used, end of segment indicators are ignored, and the following rules apply to the data transfer:

   a. If a message is the same size as the area referenced by identifier-1, the message is stored in the area referenced by identifier-1.

   b. If a message size is less than the area referenced by identifier-1, the message is aligned to the leftmost character position of the area referenced by identifier-1 with no space fill.

The SEND Statement

FUNCTION

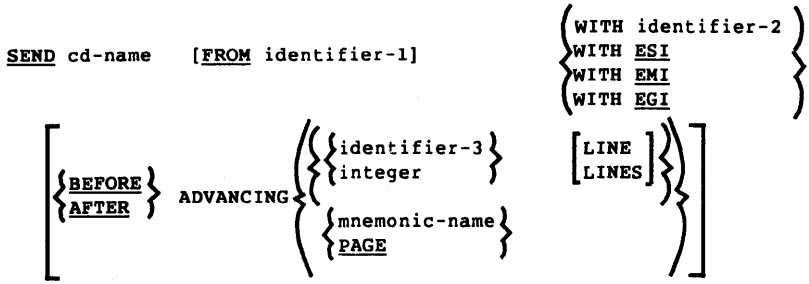
The SEND statement causes a message, a message segment, or a portion of a message or segment to be released to one or more output queues maintained by the Message Control System.

GENERAL FORMAT

Format 1:

```
SEND  cd-name FROM identifier-1
```

Format 2:



3. As part of the execution of a SEND statement, the MCS will interpret the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name to the user's indication of the number of leftmost character positions of the data item referenced by identifier-1 from which data is to be transferred.

   If the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name are zero, no characters of the data item referenced by identifier-1 are transferred.

   If the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name are outside the range of zero through the size of the data item referenced by identifier-1 inclusive, an error is indicated by the value of the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name, and no data is transferred. (See Table 13-1 for Status.)

4. As part of the execution of a SEND statement, the contents of the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name is updated by the MCS. (See **The Communication Description - Complete Entry Skeleton.**)

5. The effect of having special control characters within the contents of the data item referenced by identifier-1 is undefined.

6. A single execution of a SEND statement for Format 1 releases only a single portion of a message or of a message segment to the MCS.

   A single execution of a SEND statement of Format 2 never releases to the MCS more than a single message or a single message segment as indicated by the contents of the data item referenced by identifier-2 or by the specified indicator ESI, EMI or EGI.

   However, the MCS will not transmit any portion of a message to a communications device until the entire message is placed in the output queue.

7. During the execution of the run unit, the disposition of a portion of a message not terminated by an EMI or EGI is undefined. However, the message does not logically exist for the MCS and hence cannot be sent to a destination.

11. The hierarchy of ending indicators is EGI, EMI, ESI. An EGI need not be preceded by an ESI or EMI. An EMI need not be preceded by an ESI.

12. The ADVANCING phrase allows control of the vertical positioning of each message or message segment on a communication device where vertical positioning is applicable. If vertical positioning is not applicable on the device, the MCS will ignore the vertical positioning specified or implied.

13. If identifier-2 is specified and the content of the data item referenced by identifier-2 is zero, the ADVANCING phrase is ignored by the MCS.

14. On a device where vertical positioning is applicable and the ADVANCING phrase is not specified, automatic advancing is provided to act as if the user had specified AFTER ADVANCING 1 LINE.

15. If the ADVANCING phrase is implictly or explicitly specified and vertical positioning is applicable, the following rules apply:

    a. If identifier-3 or integer is specified, characters transmitted to the communication device will be repositioned vertically downward the number of lines equal to the value associated with the data item referenced by identifier-3 or integer.

    b. If mnemonic-name is specified, characters transmitted to the communication device will be positioned according to the rules specified for that communication device.

    c. If the BEFORE phrase is used, the message or message segment is represented on the communication device before vertical repositioning according to general rules 15a and 15b above.

    d. If the AFTER phrase is used, the message or message segment is represented on the communication device after vertical repositioning according to general rules 15a and 15b above.

    e. If PAGE is specified, characters transmitted to the communication device will be represented on the device before or after (depending upon the phrase used) the device is repositioned to the next page. If PAGE is specified but page has no meaning in conjunction with a specific device, then advancing is provided to act as if the user had specified BEFORE or AFTER (depending upon the phrase used) ADVANCING 1 LINE.

CHAPTER 14

PROGRAMMING TECHNIQUES, USEFUL HINTS, AND PROGRAM SIZING

PROGRAMMING TECHNIQUES

Although COBOL is written in an essentially free form, the user will find that a few self-imposed disciplines will be beneficial. It is suggested that these should include the following:

1. Use of the first 256 bytes of working-storage for variables which are frequently referenced will produce more compact and efficient code.

2. Use subscripts as sparingly as possible because each subscript has a storage requirement approximately equal to the size of a normal instruction.

3. For ACCEPT and DISPLAY the compiler generates one instruction per elementary item of the data-name being displayed or accepted. Therefore redefine a group of fields as a single field for DISPLAY whenever possible and avoid unnecessary numbers of small fields in an ACCEPT.

4. Use FILLER instead of a data-name for any elementary field not referenced explicitly because the word FILLER is compacted to one character in the Data Dictionary.

5. Keep the number of digits in numeric fields as small as possible.

6. Whenever possible move a group instead of several elementary moves.

The maximum number of bytes available for the user's program and work space for any given configuration, can be found in the appropriate **LEVEL II COBOL Operating Guide**. A guide for calculating the size of the program that is produced is as follows:

The sum of the record size for each file in bytes
+ the record size for each Working-Storage record in bytes
+ the number of characters in all Procedure Division literals
+ 60 bytes per file
+ 300 bytes control area
+ 6 bytes per COBOL instruction with the following qualifiers:

for an ACCEPT/DISPLAY statement add 3 bytes per elementary item within the accepted or displayed data-name.

for every subscript used in a statement add 7 bytes

for a comparison add 6 bytes

for an implicit comparison for example, PERFORM UNTIL, READ AT END, add 6 bytes

Data Dictionary

The Data Dictionary is constructed as the program is compiled. Its size depends on the host operating system. Each user-defined name will have an entry in this dictionary. The number of bytes required for each entry is given in Table 14-1.

APPENDIX A
GLOSSARY

INTRODUCTION

The terms in this Chapter are defined in accordance with their meaning as used in this document describing LEVEL II COBOL and may not have the same meaning for other languages.

These definitions are also intended to be either reference material or introductory material to be reviewed prior to reading the detailed language specifications that are contained in this manual. For this reason, these definitions are, in most instances, brief and do not include detailed syntactic rules.

Arithmetic Expression

An arithmetic expression can be an identifier or a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

Arithmetic Operator

A single character, or a fixed two-character combination, that belongs to the following set:

Character	Meaning
+	Addition
–	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Ascending Key

A key upon the values of which data is ordered starting with the lowest value of key up to the highest value of key in accordance with the rules for comparison of the data items.

Assumed Decimal Point

A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

At End Condition

A condition caused in one of three circumstances:

1. During the execution of a READ statement for a sequentially accessed file.

2. During the execution of a RETURN statement when no next logical record exists for the associated sort or merge file.

+	Plus Sign
-	Minus Sign
*	Asterisk
/	Stroke (Virgule or Slash)
=	Equal Sign
$	Currency Sign
,	Comma
;	Semicolon
.	Period (Decimal Point, Fullstop)
"	Quotation Mark
(	Left Parenthesis
)	Right Parenthesis
>	Greater Than Symbol
<	Less Than Symbol

Character Position

A character position is the amount of physical storage required to store a single standard data format character described as usage in DISPLAY.

Character-String

A sequence of contiguous characters which form a LEVEL II COBOL word, a literal, a PICTURE character-string or a comment-entry.

Class Condition

The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic or is wholly numeric.

Clause

A clause is an ordered set of consecutive LEVEL II COBOL character-strings whose purpose is to specify an attribute of an entry.

COBOL Word

(See **Word**).

Communication Device

A mechanism (hardware or hardware/software) capable of sending data to a queue and/or receiving data from a queue. This mechanism may be a computer or a peripheral device. One or more programs containing communication description entries and residing within the same computer define one or more of these mechanisms.

Communication Section

The section of the Data Division that describes the interface areas between the MCS and the program, composed of one or more CD description entries.

Compile Time

The time at which an LEVEL II COBOL source program is translated by the compiler to an LEVEL II COBOL intermediate code program.

Compiler-Directing Statement

A statement, beginning with a compiler-directing verb, that causes the compiler to take a specific action during compilation.

Complex Condition

A condition in which one or more logical operators act upon one or more conditions. (See **Negated Simple Condition, Combined Condition, Negated Combined Condition**).

Computer-Name

A system-name that identifies the computer upon which the program is to be compiled or run.

Condition

A status of a program at execution time for which a truth value can be determined. Where the term condition (condition-1, condition-2,...) appears in these language specifications in or in reference to condition (condition-1, condition-2, ...) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesised, or a negated simple condition.

Contiguous Items

Items that are described by consecutive entries in the DATA DIVISION, and that bear a definite hierarchic relationship to one another.

Counter

A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

CRT

An interactive input/output device comprising a cathode ray tube and keyboard by which an operator can enter and receive visual data.

Currency Sign

The character "$" (dollar sign) in the LEVEL II COBOL character set.

Currency Symbol

The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present in an LEVEL II COBOL source program, the currency symbol is identical to the currency sign.

Current Record

The record which is available in the record area associated with the file.

Current Record Pointer

A conceptual entity that is used in the selection of the next record from a file.

Cursor

The indicator on a CRT screen that marks the line and character position which the input/output control is currently referencing.

Declaratives

A set of one or more special purpose sections written at the beginning of the PROCEDURE DIVISION, the first of which is preceded by the key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sequence, followed by a set of associated paragraphs (0 or more).

Declarative-Sentence

A compiler-directing sentence consisting of a single USE statement terminated by the separator period (.).

Default Disk

The disk from which the compiler or Run-Time System is loaded and from which, in the absence of a specific drive identifier, any library file or called code will be loaded if required.

Delimiter

A character (or sequence of contiguous characters) that identifies the end of a string of characters, and separates that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

Descending Key

A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

Destination

The symbolic identification of the receiver of a transmission from a queue.

Editing Character

A single character or a fixed two-character combination belonging to the same set:

Character	Meaning
B	Space
0	Zero
+	Plus
–	Minus
CR	Credit
DB	Debit
Z	Zero Suppress
*	Check Protect
$	Currency Sign
,	Comma
.	Period (Decimal Point)
/	Stroke (Virgule, Slash)

Elementary Item

A data item that is described as not being further logically subdivided.

End of Procedure Division

The physical position in a LEVEL II COBOL source program after which no further procedures appear.

Entry

Any descriptive set of consecutive clauses terminated by a period (.) and written in the IDENTIFICATION DIVISION, ENVIRONMENT DIVISION or DATA DIVISION of an LEVEL II COBOL source program.

Environment Clause

A clause that appears as part of an ENVIRONMENT DIVISION entry.

File Organization

The permanent logical file structure established at the time that a file is created.

File Section

The section of the Data Division that contains file description entries together with their associated record descriptions.

Format

A specific arrangement of a set of data.

FORMS-2 Utility

An optional screen formatting utility that automatically generates LEVEL II COBOL CRT input-output coding from actual screen layout.

Group Item

A named contiguous set of elementary or group items.

High Order End

The leftmost character of a string of characters.

I-O-CONTROL

The name of an ENVIRONMENT DIVISION paragraph in which object program requirements for specific input/output techniques, rerun points, sharing of same areas by several data files, and multiple file storage on a single input/output device are specified.

I-O Mode

The state of a file after execution of an OPEN statement, with the I-O phrase specified for that file, and before the execution of a CLOSE statement for that file.

Indexed Organization

The permanent logical file structure in which each record is identified by the value of one or more keys within that record.

Indicator Area

The leftmost parameter position of a LEVEL II COBOL source record that indicates the use of the record.

Input File

A file that is opened in the input mode.

Input Mode

The state of a file after execution of an OPEN statement, with the INPUT phrase specified for that file, and before the execution of a CLOSE statement for that file.

Input-Output File

A file that is opened in the I-O mode.

Input-Output Section

The section of the ENVIRONMENT DIVISION that names the files and the external media used by a program and which provides information required for transmission and handling of data during execution of the run-time program.

Input Procedure

A set of statements that is executed each time a record is released to the sort file.

Integer

A numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point. Where integer appears in general formats, integer must not be a numeric data item, and must not be signed, nor zero unless explicitly allowed by the rules of that format.

Level-Number

A user-defined word which indicates the position of a data item in the hierarchical structure of a logical record or which indicates special properties of a data description entry. A level-number is expressed as a one or two digit number. Level-numbers in the range 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record.

Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-numbers 66, 77 and 88 identify special properties of a data description entry.

Library-Name

A user-defined word that names a LEVEL II COBOL library source file that is to be used by the compiler for a given source program compilation.

Library-Text

A sequence of character-strings and/or separators in a COBOL library.

Line Sequential File Organization

A sequential file containing variable length records separated by the CR (carriage return) and LF (line feed) characters.

Linkage Section

The section in the DATA DIVISION of the called program that describes data items available from the calling program. These data items may be referred to by both the calling and called program.

Literal

A character-string whose value is implied by the ordered set of characters comprising the string.

Message Indicators

EGI (end of group indicator), EMI (end of message indicator), and ESI (end of segment indicator) are conceptual indications that serve to notify the MCS that a specific condition exists (end of group, end of message, end of segment). Within the hierarchy of EGI, EMI, and ESI, an EGI is conceptually equivalent to an ESI, EMI, and EGI. An EMI is conceptually equivalent to an ESI and EMI. Thus, a segment may be terminated by an ESI, EMI, or EGI. A message may be terminated by an EMI or EGI.

Message Segment

Data that forms a logical subdivision of a message normally associated with an end of segment indicator. (See **Message Indicators**).

Mnemonic-Name

A user-defined word that is associated in the ENVIRONMENT DIVISION with a specified implementor-name.

Native Character Set

The implementor-defined character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

Native Collating Sequence

The implementor-defined collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

Negated Combined Condition

The 'NOT' logical operator immediately followed by a parenthesized combined condition.

Negated Simple Condition

The 'NOT' logical operator immediately followed by a simple condition.

Numeric Item

A data item whose description restricts its contents to a value represented by characters chosen from the digits '0' through '9'; if signed, the item may also contain a '+', '–', or other representation of an operational sign.

Numeric Literal

A literal composed of one or more numeric characters that also may contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

Object-Computer

The name of an ENVIRONMENT DIVISION paragraph in which the computer environment, within which the run-time program is executed, is described.

Open Mode

The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O or EXTEND.

Operand

Whereas the general definition of operand is 'that component which is operated upon', for the purposes of this publication, any lowercase word (or words) that appears in a statement or entry format may be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

Operational Sign

An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

Optional Word

A reserved word that is included in a specified format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

In the ENVIRONMENT DIVISION:

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
FILE-CONTROL.
I-O-CONTROL.
```

Paragraph-Name

A user-defined word that identifies and begins a paragraph in the PROCEDURE DIVISION.

Phrase

A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a LEVEL II COBOL procedural statement or of a COBOL clause.

Physical Record

(See **Block**)

Prime Record Key

A key whose contents uniquely identify a record within an indexed file.

Procedure

A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the PROCEDURE DIVISION.

Procedure-Name

A user-defined word which is used to name a paragraph or section in the PROCEDURE DIVISION. It consists of a paragraph-name or a section-name.

Program-Name

A user-defined word that identifies a COBOL source program.

Queue Name

A symbolic name that indicates to the MCS (Message Control System) the logical path by which a message or a portion of a completed message may be accessible in a queue.

Random Access

An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from or placed into a relative or indexed file.

Record

(see **Logical Record**)

Record Area

A storage area allocated for the purpose of processing the record described in a record description entry in the File Section.

Record Description

(See **Record Description Entry**)

Record Description Entry

The total set of data description entries associated with a particular record.

Record Key

A key, either the prime record key or an alternate record key, whose contents identify a record within an indexed file.

Record-Name

A user-defined word that names a record described in a record description entry in the DATA DIVISION.

Relative File

A file with relative organization.

Relative Key

A key whose contents identify a logical record in a relative file.

Relative Organization

The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

Reserved Word

A COBOL word specified in the list of words which may be used in COBOL source programs, but which must not appear in the programs as user-defined words or system-names.

Routine-Name

A user-defined word that identifies a procedure written in a language other than COBOL

Run-Time

The time at which the intermediate code produced by the compiler is interpreted by the Run-Time System for execution.

Run-Time System (RTS)

The software that interprets the intermediate code produced by the LEVEL II COBOL compiler and enables it to be executed by providing interfaces to the operating system and CRT.

Segment-Number

A user-defined word which classifies sections in the PROCEDURE DIVISION for purposes of segmentation. Segment-numbers may contain only the characters '0', '1', ..., '9'. A segment-number may be expressed either as a one or two digit number, and is checked for syntax only.

Sentence

A sequence of one or more statements, the last of which is terminated by a period followed by a space.

Separator

A punctuation character used to delimit character-strings.

Sequential Access

An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of records in the file.

Sequential File

A file with sequential organization.

Sequential Organization

The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

Sign Condition

The proposition, for which a truth value can be determined, that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

Special Character

A character that belongs to the following set:

Character	Meaning
+	plus sign
–	minus sign
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol

Special-Character Word

A reserved word which is an arithmetic operator or a relation character.

SPECIAL-NAMES

The name of an ENVIRONMENT DIVISION paragraph in which implementor-names are related to user-specified mnemonic-names.

Special Registers

Compiler generated storage areas whose primary use is to store information produced in conjunction with the user of specified COBOL features.

Symbol Function

The use of specified characters in the PICTURE clause to represent data types.

System-Name

A COBOL word which is used to communicate with the operating environment.

Syntax

The order in which elements must be put together to form a program.

Table

A set of logically consecutive items of data that are defined in the DATA DIVISION by means of the OCCURS clause.

Table Element

A data item that belongs to the set of repeated items comprising a table.

Terminal

The originator of a transmission to a queue, or the receiver of a transmission from a queue.

Text-Name

A user-defined word which identifies library text.

Text-Word

Any character-string or separator, except space, in a COBOL library or in pseudo-text.

Truth Value

The representation of the result of the evaluation of a condition in terms of one of two values

true
false

APPENDIX B

IBM EXTENSIONS

The following IBM extensions are implemented in the full LEVEL II COBOL product:

1. "ID" is a synonym for "IDENTIFICATION".

2. SKIP*n* where *n* is 1, 2, or 3 causes the specified number of blank lines to be inserted, SKIP*n* can occur anywhere in the source program and causes the specified number of lines to be skipped after this statement.

3. EJECT causes a new page to be thrown. EJECT can occur anywhere in the source program and causes the output that follows to start on a new page.

4. GOBACK causes execution to return to the calling program, or to the operating system if the program containing this verb is not a called program. This verb is the equivalent of the LEVEL II COBOL syntax:

```
EXIT PROGRAM.
STOP RUN.
```

5. A PASSWORD can be associated with a file in the SELECT statement.

6. ENTRY literal USING data-name-1 [, data-name-2] ... can be used to define an alternative entry point to a subprogram. This syntax is equivalent to the LEVEL II COBOL phrase PROCEDURE DIVISION USING. The ENTRY phrase is for documentary purposes only.

These LEVEL II COBOL extensions are allowed only if the IBM directive is set on the compiler command line (see the **LEVEL II COBOL Operating Guide**).

APPENDIX C

COMMUNICATION FACILITY - CONCEPTS

The communication facility provides the ability to access, process, and create messages or portions thereof. It provides the ability to communicate through a Message Control System with local and remote communication devices.

THE MESSAGE CONTROL SYSTEM

The implementation of the communication facility requires that a Message Control System (MCS) be present in the COBOL object program's operating environment.

The MCS is the logical interface to the operating system under which the COBOL object program operates. The primary functions of the MCS are the following:

1. To act as an interface between the COBOL object program and the network of communication devices, in much the same manner as an operating system acts as an interface between the COBOL object program and such devices as card readers, magnetic tape and mass storage devices, and printers.

2. To perform line discipline, including such tasks as dial-up, polling, and synchronization.

3. To perform device-dependent tasks, such as character translation and insertion of control characters, so that the COBOL user can create device-dependent programs.

2. The SEND statement, which causes data associated with the COBOL object program to be passed to one or more queues, and;

3. The ACCEPT statement with the COUNT phrase, which causes the MCS to indicate to the COBOL object program the number of complete messages in the specified queue structure.

The COBOL source program uses two statements to control the interface between the MCS and the communication devices:

1. The ENABLE statement, which establishes logical connection between the MCS and one or more given communication devices, and;

2. The DISABLE statement, which breaks a logical connection between the MCS and one or more given communication devices.

These relationships are shown in Figure C-1, COBOL Communication Environment.

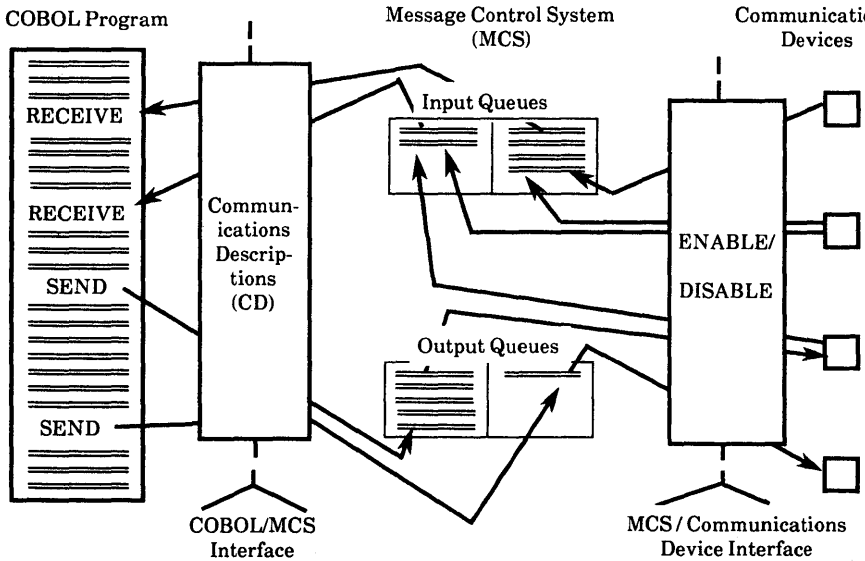


Figure C-1. COBOL Communication Environment

DETERMINING THE METHOD OF SCHEDULING

A COBOL source program can be written so that its object program can operate with either of the two modes of scheduling. In order to determine which method was used to load the COBOL object program, the following is one technique that may be used:

1. One CD must contain a FOR INITIAL INPUT clause.

2. The PROCEDURE DIVISION may contain statements to test the initial value of the symbolic queue name in that CD. If it is space-filled, job control statements were used to schedule the COBOL object program. If not space filled, the MCS has invoked the COBOL object program and replaced the spaces with the symbolic name of the queue containing the message to be processed.

THE CONCEPT OF MESSAGES AND MESSAGE SEGMENTS

A message consists of some arbitrary amount of information, usually character data, whose beginning and end are defined or implied. As such, messages comprise the fundamental but not necessarily the most elementary unit of data to be processed in a COBOL communication environment.

Messages may be logically subdivided into smaller units of data called message segments which are delimited within a message by means of end of segment indicators (ESI). A message consisting of one or more segments is delimited from the next message by means of an end of message indicator (EMI). In a similar manner, a group of several messages may be logically separated from succeeding messages by means of an end of group indicator (EGI). When a message or message segment is received by the COBOL program, a communication description interface area is updated by the MCS to indicate which, if any, delimiter was associated with the text transferred during the execution of that RECEIVE statement. On output the delimiter, if any, to be associated with the text released to the MCS during execution of a SEND statement is specified or referenced in the SEND statement. Thus the presence of these logical indicators is recognized and specified both by the MCS and by the COBOL object program; however, no indicators are included in the message text processed by COBOL programs.

2. When the COBOL object program creates output messages at a speed faster than the destination can receive them.

Enabling and Disabling Logical Connectives

Usually, the MCS will logically connect and disconnect sources and destinations based on time of day, message activity, or other factors unrelated to the COBOL program. However, the COBOL program has the ability to perform these functions through use of the ENABLE and DISABLE statements.

A key is required in both statements in order to prevent indiscriminate use of the facility by a COBOL user who is not aware of the total network environment, and who may therefore disrupt system functions by the untimely issuance of ENABLE and DISABLE statements. However, this action never interrupts a transmission.

Enqueueing and Dequeueing Methods

In systems that allow the user to specify certain MCS functions, it may be necessary that the user specify to the MCS, prior to execution of programs which reference these facilities, the selection algorithm and other designated MCS functions to be used by the MCS in placing messages in the various queues. A typical selection algorithm, for example, would specify that all messages from a given source be placed in a given input queue, or that all messages to be sent to a given destination be placed in a given output queue.

Dequeueing is often done on a first in, first out basis. Thus, messages dequeued from either an input or output queue are those messages which have been in the queue for the longest period of time. However, the MCS can, upon prior specification by the user, dequeue on some other basis, that is, priority queueing can be employed.

The following examples illustrate the effect of the algorithms shown in Figure C-2:

1. The program executes a RECEIVE statement, specifying via the CD:

   Queue A
   MCS returns: Message Z1

2. The program executes a RECEIVE statement, specifying via the CD:

   Queue A
   sub-queue-1 C
   MCS returns: Message Y7

3. The program executes a RECEIVE statement, specifying via the CD:

   Queue A
   Sub-queue-1 B
   Sub-queue-2 E
   MCS returns: Message X1

4. The program executes a RECEIVE statement, specifying via the CD:

   Queue A
   Sub-queue-1 C
   Sub-queue-2 G
   Sub-queue-3 N
   MCS returns: Message X6

To access the next message in a queue, regardless of which sub-queue that message may be in, only the queue name need be specified. The MCS, when supplying the message, will return to the COBOL object program any applicable sub-queue names via the data items in the associated CD. If, however, the next message in a given sub-queue is desired, both the queue name and any applicable sub-queue names must be specified.

For output, the COBOL user specifies only the destination(s) of the message, and the MCS places the message in the proper output queue structure.

There is no one-to-one relationship between a communication device and a source/destination. A source or destination may consist of one or more physical devices. The device or devices which comprise a source/destination are defined to the MCS.

F

Q

R

S

Z

USER'S COMMENT SHEET

High Performance LEVEL II COBOL™
Language Rererence, First Edition
09-01991-01

We welcome your comments and suggestions. They help us improve our manuals. Please give specific page and paragraph references whenever possible.

Does this manual provide the information you need? Is it at the right level? What other types of manuals are needed?

Is this manual written clearly? What is unclear?

Is the format of this manual convenient in arrangement, in size?

Is this manual accurate? What is inaccurate?

Name _____ Date _____

Title _____ Phone _____

Company Name/Department _____

Address _____

City _____ State _____ Zip Code _____

Thank you. All comments become the property of Convergent Technologies, Inc.