CIS COBOL Language Reference Manual

Version 4.5

CIS COBOL Language Reference Manual: Version 4.5

The product described in this manual and products for use with it are subject to continuous development and improvement. All information of a technical nature and particulars of the product and its use (including the information and particulars in this manual) are given by Acorn Computers in good faith. However, it is acknowledged that there may be errors or omissions in this manual. A list of details of any amendments or revisions to this manual can be obtained upon request from Acorn Computers Technical Enquiries. Acorn Computers welcome comments and suggestions relating to the product and this manual.

All correspondence should be addressed to:

Micro Focus Limited
26 West street
Newbury Berkshire
RG13 1JT

Acknowledgements

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL Programming Language Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection herewith.

The authors and copyright holders of the copyrighted material used herein:

FLOW-MATIC (Trademark for Sperry Rand Corporation) Programming for the Univac® I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

Table of Contents

List of Figures

List of Tables

PREFACE

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

Each release of CIS COBOL is characterized by a two-digit code in the form of

"Version number". "Release number within 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 11 apply to all users and describe additional features and facilities available with the standard language. The appendices supply reference information pertinent to all systems.

The manual contains the following chapters and appendices:

"Chapter 1. Introduction", which gives a general description of the language, including a broad outline of ANSI COBOL features included and omitted and additional features of CIS COBOL.

"Chapter 2. COBOL Concepts", which describes general concepts of the COBOL language including program structure, and details of statement components and notation.

"Chapter 3. Nucleus", which describes the nucleus of all COBOL programs and the layout of each program division in the nucleus.

"Chapter 4. Table Handling", which describes the handling of data tables in the Data and Procedure divisions of a COBOL program.

"Chapter 5. Sequential Input and Output", which describes the programming of input and output of data in files with sequential format.

"Chapter 6. Relative Input and Output", which describes the programming of input and output of data in files with relative format.

"Chapter 7. Indexed Input and Output", which describes the programming of input and output of data in files with indexed format.

"Chapter 8. Segmentation", which describes the facility for specifying permanent and independent object program segments.

"Chapter 9. Library", which describes the source library maintenance feature of COBOL.

"Chapter 10. Debug and Interactive Debugging", which describes the basic and interactive debugging features available in CIS COBOL.

"Chapter 11. Interprogram Communication", which describes the ability of CIS COBOL programs to interface during running and to access common data, enabling modular programming.

"Chapter 12. Programming Techniques and Sizing", which describes the means available for CIS COBOL programmers to estimate object program size and includes programming techniques in CIS COBOL.

"Appendix A. Reserved Word Table", which lists words reserved for CIS COBOL functions within a program.

"Appendix B. Character Set and Collating Sequence", which lists all characters available and their collating sequence.

"Appendix C. Glossary", which lists specific terms used in CIS COBOL.

"Appendix D. Compile - Time Errors", which lists all errors that can be signalled during program compilation.

"Appendix E. Run-Time Errors", which lists all errors that can be signalled during program execution.

"Appendix F. Syntax Summary", which summarizes the syntax used in CIS COBOL programming.

"Appendix G. Summary of Extensions to ANSI COBOL", which summarizes all extensions to ANSI COBOL provided by CIS COBOL.

"Appendix H. System Dependent Language Features", which describes the system dependent CIS COBOL entries for use with microcomputers and those features not included because of hardware requirements.

"Appendix I. Language Specification", which is an overall specification of the CIS COBOL language.

RELATED PUBLICATIONS

No discussion of operating the CIS COBOL Compiler or Run-Time system is incorporated in this manual. Please refer to document:

CIS COBOL Operating Guide (for use with the relevant Operating System)

NOTATION IN THIS MANUAL

Throughout this manual, the following notation is used to describe the format of COBOL statements:

1. All words printed in capital letters which are underlined must always be present when the functions of which they are a part are used. An error printout will occur during compilation if the underlined words are absent or incorrectly spelled. The underlining is not necessary when writing a COBOL source program.

2. All words printed in capital letters which are not underlined are used for readability only. They may be written, or not, as the programmer wishes.

3. All words printed in small letters are generic terms representing names which will be devised by the programmer.

4. When material is enclosed in braces { } , a choice must be made from the options within them.

5. When material is enclosed in square brackets [ ], it is an indication that the material is an option which may be included or omitted as required.

6. When material is enclosed in square brackets crossed [̶]̶, it is an indication that the material is mandatory when the ANSI switch is set (see Chapter 2) but optional otherwise.

7. Language features that are shaded in the text are language extensions which exceed the ANSI standard.

8. In text, the ellipsis (...) shows the omission of a portion of a source program or a sequence. This meaning becomes apparent in context.

   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.

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

CHAPTER N Chapter Heading TITLE ORDER ONE HEADING ORDER TWO HEADING Order Three Heading Text two lines down Order Four Heading Order Five Heading: Text on same line

Numbers one (1) to nine (9) are written in text as letters, e.g. one.

Numbers ten (10) upwards are written in text as numbers, e.g. 12.

The phrase "For documentation purposes only" in the text of this manual means that the associated coding is accepted syntactically by the Compiler, but is ignored when producing the object program.

Chapter 1. Introduction

WHAT IS CIS COBOL?

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

CIS 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 given in "American National Standard Programming Language COBOL" (ANSI X3.23 1974). The CIS COBOL implementation has been selected from both levels of ANSI COBOL. The following modules are fully implemented at Level 1:

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

In addition many Level 2 features are implemented such as:

- Nucleus - Nested IF, PERFORM UNTIL
- Relative and Indexed sequential I/O - START statement
- Inter-Program Communication - CANCEL statement

This manual is intended as a reference work for COBOL programmers and material from the ANSI 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 Compiler Testing Center (FCTC) under the direction of the General Services Administration (GSA) as validated at Federal Low Intermediate Level. The GSA Validation Summary Report is available under the reference CCVS74-VSR685.

Along with the ANSI implementation CIS COBOL also contains several language extensions specifically oriented to the small computer environment. These enable a CIS COBOL program to format CRT screens for data input and output (DISPLAY and ACCEPT), READ and WRITE text files efficiently and define external file names at run time.

The programmer wishing to transport an existing COBOL program to run under CIS COBOL must check that the individual language features he has used are supported by CIS COBOL. The COBOL SECTION statements in the Segmentation feature can be performed using the PERFORM statement.

A compile time ANSI switch can be set that makes certain COBOL source mandatory, whereas if not set it is optional. (See Chapter 2).

The CIS COBOL compiler is designed to enable programs to be developed in a 48K machine. The Compiler supports sequential, relative and indexed sequential files, as well as interactive communications via the ACCEPT and DISPLAY verbs.

The CIS COBOL System also contains a powerful utility called FORMS-2 that enables the Operator to define screen layouts from a screen “module” and produce automatically the data description for direct inclusion in a CIS COBOL program. This is described in the *CIS COBOL Operating Guide*.

CIS 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 CIS COBOL program is provided by the Compiler during compilation. Error messages are inserted in the listing. Interactive Debugging facilities are provided for run-time use, and these are described in the *CIS COBOL Operating Guide*.

CIS 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 Operating Guide.

PROGRAM STRUCTURE

A 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.

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, i.e., the order of the elements of the statement and restrictions on what each element may represent.

GENERAL RULES

A general rule is a rule that defines or clarifies the meaning or relationship of meanings of an element or set of elements. It is used to define or clarify the semantics of the statement and the effect that it has on either execution or compilation.

ELEMENTS

Elements which make up a clause or a statement consist of uppercase words, lowercase words, level-numbers, brackets, braces, connectives and special characters (see Chapter 2).

SOURCE FORMAT

The COBOL source format divides each COBOL source record into 72 columns. These columns are used in the following way:

Columns 1 - 6	Sequence number
Column 7	Indicator area
Columns 8 - 11	Area A
Columns 12 -72	Area B

SEQUENCE NUMBER

A sequence number of six digits may be used to identify each source program line.

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 line.

Section names and paragraph names begin in Area A and are followed by a period and a space. Level indications FD, 01 and 77 begin in Area A and are followed in Area B by the appropriate file and record description.

Program sentences may commence anywhere in Area A or Area B. More than one sentence is permitted in each source record.

Note that TAB characters are not permitted in CIS COBOL source.

Figure 1-1 shows the source format of a typical program.

```
** CIS COBOL V4.5                    STOCK.CBL                      PAGE: 0001
**
** OPTIONS SELECTED :
**     FORM(72)
**
000010 IDENTIFICATION DIVISION.                                          0118
000020 PROGRAM-ID. STOCK-FILE-SET-UP.                                    0118
000030 AUTHOR. MICRO FOCUS LTD.                                          0118
000040 ENVIRONMENT DIVISION.                                             0118
000050 CONFIGURATION SECTION.                                            0118
000060 SOURCE-COMPUTER.                                                  0118
000070 OBJECT-COMPUTER.                                                  0118
000075 SPECIAL-NAMES. CONSOLE IS CRT.                                    0118
000080 INPUT-OUTPUT SECTION.                                             0118
000090 FILE-CONTROL.                                                     0118
000100     SELECT STOCK-FILE ASSIGN "STOCK.IT"                           0184
000110     ORGANIZATION INDEXED                                          0186
```

```
000120     ACCESS DYNAMIC                                               0186
000130     RECORD KEY STOCK-CODE.                                       0186
000140 DATA DIVISION.                                                   01BE
000150 FILE SECTION.                                                    01BE
000160 FD  STOCK-FILE: RECORD 32.                                       01BE
000170 01  STOCK-ITEM.                                                  01BE
000180     02 STOCK-CODE PIC X(4).                                      01BE
000190     02 PRODUCT-DESC PIC X(20).                                   01C2
000200     02 UNIT-SIZE PIC 9(4).                                       01D6
000210 WORKING STORAGE SECTION.                                         01DC
000220 01  SCREEN-HEADINGS.                                             01DC 00
000230     02  ASK-CODE PIC X(21) VALUE "STOCK CODE     <    >".       01DC 00
000240     02  FILLER PIC X(59).                                        01F3 15
000250     02  ASK-DESC PIC X(16) VALUE "DESCRIPTION    <".            022C 50
000260     02  SI-DESC PIC X(21) VALUE "                   >".        023C 60
000270     02  FILLER PIC(43).                                          0251 75
000280     02  ASK-SIZE PIC X(21) VALUE "UNIT SIZE      <    >".       027C A0
000290  01  ENTER-IT REDEFINES SCREEN-HEADINGS.                         01DC 00
000300     02  FILLER PIC X(16).                                        01DC 00
000310     02  CRT-STOCK-CODE   PIC X(4).                               01EC 10
000320     02  FILLER       PIC X(76).                                  01F0 14
000330     02  CRT-PROD-DESC    PIC X(20).                              023C 60
000340     02  FILLER       PIC X(60).                                  0250 74
000350     02  CRT-UNIT-SIZE    PIC 9(4).                               028C B0
000360     02  FILLER           PIC X.                                  0290 B4
000370 PROCEDURE DIVISION.                                              0000
000380 SR1.                                                             001C 00
000390     DISPLAY SPACE.                                               001D
000400     OPEN I-O STOCK-FILE.                                         0020
000410     DISPLAY SCREEN-HEADINGS.                                     0024
000420 NORMAL-INPUT.                                                    0038 00
000430     MOVE SPACE TO ENTER-IT.                                      0039
000440     DISPLAY ENTER-IT.                                            003F
000450 CORRECT-ERROR.                                                   0056 00
000460     ACCEPT ENTER-IT.                                             0057
000470     IF CRT-STOCK-CODE = SPACE GO TO END-IT.                      006E
000480     IF CRT-UNIT-SIZE NOT NUMERIC GO TO CORRECT-ERROR.            0078
000490     MOVE CRT-PROD-DESC TO PRODUCT-DESC.                          0081
000500     MOVE CRT-UNIT-SIZE TO UNIT-SIZE.                             0087
000510     MOVE CRT-STOCK-CODE TO STOCK-CODE.                           008F
000520     WRITE STOCK-ITEM INVALID KEY GO TO CORRECT-ERROR.            0095
000530     GO TO NORMAL-INPUT.                                          00A1
000540 END-IT.                                                          00A4 00
000550     CLOSE STOCK-FILE.                                            00A5
000560     DISPLAY SPACE.                                               00A9
000570     DISPLAY "END OF PROGRAM".                                    00AC
000580     STOP RUN.                                                    00BD
** CIS COBOL V4.5 REVISION 4                                 URN AA/0000/AA
** COMPILER COPYRIGHT (C) 1978,1982 MICRO FOCUS LTD
** ERRORS=00000 DATA=00768 CODE=00256 DICT=00409:20662/21071 GSA FLAGS=  OFF
<---->|<-><------------------------------------------------------------><---->
  |   | |     |                                      Inserted by Compiler---+
  |   | |     +-- Columns 12-72 - Area B
  |   | +-- Columns 8-11 - Area A
  |   +-- Column 7 - Indicator Area
  +-- Columns 1-6 - Sequence Number
```

Figure 1.1. Sample Program Listing Showing Source Format

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 CIS 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 CIS 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 Appendix B.

LANGUAGE STRUCTURE

The individual characters of the language are concatenated to form character-strings and separators. A separator may be concatenated with another separator or with a character-string. A character-string may only be concatenated with a separator. The concatenation of character-strings and separators forms the text of a source program.

Separators

A separator is a string of one or more punctuation characters. The rules for formation of separators are:

1. The punctuation character space is a separator. Anywhere a space is used as a separator, more than one space may be used.

2. The punctuation characters comma, semicolon and period, when immediately followed by a space, are separators. These separators may appear in a COBOL source program only where explicitly permitted by the general formats, by format punctuation rules (see FORMATS AND

RULES in Chapter 1), by statement and sentence structure definitions (see STATEMENTS AND SENTENCES in this Chapter), or reference format rules (see REFERENCE FORMAT in this Chapter).

3. The punctuation characters right and left parenthesis are separators. Parenthesis may appear only in balanced pairs of left and right parentheses delimiting subscripts, indices, arithmetic expressions, or conditions.

4. The punctuation character quotation mark is a separator. An opening quotation mark must be immediately preceded by a space or left parenthesis; a closing quotation mark must be immediately followed by one of the separators space, comma, semicolon, period, or right parenthesis.

   Quotation marks may appear only in balanced pairs delimiting nonnumeric literals except when the literal is continued. (See CONTINUATION OF LINES in this Chapter).

5. 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.

6. 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 CIS COBOL word, a literal, a PICTURE character-string, or a comment-entry. A character-string is delimited by separators.

COBOL Words

A COBOL word is a character-string of not more than 30 characters which forms a user defined word, a system-name, or a reserved word. Within a given source program these classes form disjoint sets; a COBOL word may belong to one and only one of these classes.

User-Defined Words: A user-defined word is a COBOL word that must be supplied by the user to satisfy the format of a clause or statement. Each character of a user-defined word is selected from the set of characters 'A', 'B', 'C', ... 'Z', 'a', 'b', 'c', ... 'z', '0', ... '9', and '-', except that the '-' may not appear as the first or last character. The exception to this rule is an external file-name-literal which must be a normal alphanumeric literal.

User-defined word types which are implemented are as follows:

alphabet-name
condition-name
data-name
external-file-name-literal
file-name
index-name

level-number
mnemonic-name
paragraph-name
program-name
record-name
section-name
segment-number
text-name

Within a given source program, ten of these 12 types of user-defined words are grouped into nine disjoint sets. The disjoint sets are:

alphabet-names
condition-names, data-names, and record-names
file-names
index-names
mnemonic-names
paragraph-names
program-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).

Paragraph-Name: A paragraph-name is a word which names a paragraph in the Procedure Division. Paragraph-names are equivalent if, and only if, they are composed of the same sequence of the same number of digits and/or characters.

Section-Name: A section-name is a word which names a section in the Procedure Division. Section names are equivalent if, and only if, they are composed of the same sequence of the same number of digits and/or characters.

Other User-Defined Names: See the glossary in Appendix C for definitions of all other types of user-defined words.

System-Names: A system-name is a COBOL word which is used to communicate with the operating environment. Each character used in the formation of a system-name must be selected from the set of characters 'A', 'B' , 'C', ... 'Z', 'a','b', ... 'z', '0' ... '9' and'-', except that the '- ' may not appear as the first or last character.

There are three types of system-names:
1. computer-name
2. implementor-name
3. language-name

Within a given implementation these three types of system-names form disjoint sets; a given system-name may belong to one and only one of them. The system-names listed above are individually defined in the glossary in Appendix C.

Reserved Words: A reserved word is a COBOL word that is one of a specified 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. Reserved words can only be used as specified in the general formats. (See Appendix A).

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

Figurative Constants: Certain reserved words are used to name and reference specific constant values. These reserved words are specified under Figurative Constant Values in this chapter.

Literals

A literal is a character-string whose value is implied by an ordered set of characters of which the literal is composed or by specification of a reserved word which references a figurative constant. Every literal belongs to one of two types, nonnumeric or numeric.

Nonnumeric Literals A nonnumeric literal is a character-string delimited on both ends by quotation marks and consisting of any allowable character in the computer's character set. Allowed are nonnumeric literals of 1 through 128 characters in length. To represent a single quotation mark character

within a nonnumeric literal, two contiguous quotation marks must be used. The value of a nonnumeric literal in the object program is the string of characters itself, except:

1. The delimiting quotation marks are excluded, and
2. Each embedded pair of contiguous quotation marks represents a single quotation mark character.

All other punctuation characters are part of the value of the nonnumeric literal rather than separators; all nonnumeric literal are category alphanumeric. (See the section called “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 hex 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 quality represented by the characters in the numeric literal. Every numeric literal is category numeric. (See the section called “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.

Figurative Constant Values

Figurative Constant Values are generated by the compiler and referenced through the use of the reserved words given below. These words must not be bounded by quotation marks when used as figurative constants. The singular and plural forms of figurative constants are equivalent and may be used interchangeably.

The figurative constant values and the reserved words used to reference them are shown in Table 2-1.

Table 2.1. Figurative Constants and their Reserved Words

CONSTANT	REPRESENTATION
ZERO ZEROS ZEROES	Represents the value '0', or one or more of the character '0' depending on context.
SPACE SPACES	Represents one or more of the character space from the computer's character set.

CONSTANT	REPRESENTATION
HIGH-VALUE HIGH-VALUES	Represents one or more of the character that has the highest ordinal position in the program collating sequence.
LOW-VALUE LOW-VALUES	Represents one or more of the character that has the lowest ordinal position in the program collating sequence.
QUOTE QUOTES	Represents one or more of the character ' " '. The word QUOTE or QUOTES cannot be used in place of a quotation mark in a source program to bound a nonnumeric literal. Thus, QUOTE ABD QUOTE is incorrect as a way of stating the nonnumeric literal "ABD".
ALL literal	Represents one of more repetitions of the single character comprising the literal (literal may not be more than one character in length). The literal must be either a nonnumeric literal or a figurative constant other than ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readability only.

When a figurative constant represents a string of one or more characters, the length of the string i s 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 or STOP statement, the length of the string is one character. DISPLAY SPACE 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 section called “THE PICTURE CLAUSE” for the PICTURE character-string and for the rules that govern their use.

Any punctuation character which appears as part of the specification of a PICTURE character-string is not considered as a punctuation character, but rather as a symbol used in the specification of that PICTURE character-string.

Comment-Entries

A comment-entry is an entry in the Identification Division that may be any combination of characters from the computer's character set.

CONCEPT OF COMPUTER INDEPENDENT DATA DESCRIPTION

To make data as computer independent as possible, the characteristics or properties of the data are described in relation to a standard data format rather than an equipment-oriented format. This standard data format is oriented to general data processing applications and uses the decimal system to represent numbers (regardless of the radix used by the computer) and the remaining characters in the CIS COBOL character set to describe nonnumeric data items.

Concept of Levels

A level concept is inherent in the structure of a logical record. This concept arises from the need to specify subdivisions of a record for the purpose of data reference. Once a subdivision has been specified, it may be further subdivided to permit more detailed data referral.

The most basic subdivisions of a record, that is, those not further subdivided, are called elementary items; consequently, a record is said to consist of a sequence of elementary items, or the record itself may be an elementary item.

In order to refer to a set of elementary items, the elementary items are combined into groups. Each group consists of a named sequence of one or more elementary items. Groups, in turn, may be combined into groups of two or more groups, etc. Thus, an elementary item may belong to more than one group.

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 is a special level-number, 77, which is an exception 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 group items must not exceed 8192 Bytes in length.

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

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

2. Entries that specify condition-names.

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.

Concept of Classes of Data

The five categories of data items (see the section called “THE PICTURE CLAUSE” in Chapter 3) are grouped into three classes : alphabetic, numeric, and alphanumeric. For alphabetic and numeric, the classes and categories are synonymous. The alphanumeric class includes the categories of alphanumeric edited, numeric edited and alphanumeric (without editing). Every elementary item except for an index data item belongs to one of the classes and to one of the categories. The class of a group item is treated at object time as alphanumeric regardless of the class of elementary items

subordinate to that group item. Table 2-2 depicts the relationship of the class and categories of data items.

Table 2.2. Data Levels, classes and categories

LEVEL OF ITEM	CLASS	CATEGORY
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric Edited Alphanumeric Edited Alphanumeric
Non-Elementary Group	Alphanumeric	Alphabetic Numeric Numeric Edited Alphanumeric Edited Alphanumeric

Selection of Character Representation and Radix

The value of a numeric item may be represented in either binary or decimal form, depending on the equipment. In addition, there are several ways of expressing decimal. Since these representations are actually combinations of bits, they are commonly called binary-coded decimal forms. The four standard formats used for storing numeric data in CIS COBOL are as follows:

1. As alphanumeric characters stored one per byte in ASCII representation.

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 section called “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	12-14
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.

   EXAMPLE:

   a. For COMPUTATIONAL-3 and PICTURE 9999, the number +1234 would be stored as follows:

```
...      0     1     2      3      4      F
       0000  0001  0010   0011   0100   1111
                                 +---------+
                                   1 byte
```

      where F represents the non-printing plus sign.

   b. For COMPUTATIONAL-3 and PICTURE S9999, the number +1234 would be stored as follows:

      Storage would be as in a above except that the least significant digit would be replaced by C (1100) representing the plus sign.

   c. For COMPUTATIONAL-3 and PICTURE S9999, the number -1234 would be stored as follows:

      Storage would be as in a above except that the least significant byte would be replaced by D (1101) representing the minus sign.

   Table 2-4 shows the storage requirements for each COMP-3 clause.

Table 2.4. Numeric Data Storage for the COMPUTATION-3 PICTURE Clause.

Bytes Required	Number of Digits (Signed or Unsigned)
1	1
2	2-3
3	4-5
4	6-7
5	8-9
6	10-11
7	12-13
8	14-15
9	16-17
10	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.

Uniqueness of Reference

Subscripting

Subscripts can be used only when reference is made to an individual element within a list or table of like elements that have not been assigned individual data-names (see THE OCCURS CLAUSE in Chapter 4). The subscript can be represented either by a numeric literal that is an integer or by a data-name, The data-name must be a numeric elementary item that represents an integer. The subscript may be signed and, if signed, it must be positive. The lowest possible subscript value is 1. This value points to the first element of the table. The next sequential elements of the table are pointed to by subscripts whose values are 2, 3, ... The highest permissible subscript value, in any particular case, is the maximum number of occurrences of the item as specified in the OCCURS clause.

The subscript, or set of subscripts, that identifies the table element is delimited by the balanced pair of separators left parenthesis and right parenthesis following the table element data-name. The table element data-name appended with a subscript is called a subscripted data-name or an identifier. When more than one subscript is required, they are written in the order of successively less inclusive dimensions of the data organization.

The format is:

data-name (subscript-1 [, subscript-2 [, subscript-3]])

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. 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.

The general format for indexing is:

{ *data-name* | *condition-name* } ({ *index-name-1* | *literal-1* | [{ + | - } *literal-2*]}
[, { *index-name-2* | *literal-3* } [{ + | - | *literal-4* }] [, { *index-name-3* | *literal-5* | [{ + | - } *literal-6*]}]])

Identifier

An identifier is a term used to reflect that a data-name, if not unique in a program, must be followed by a syntactically correct combination of subscripts or indices necessary to ensure uniqueness.

The general formats for identifiers are:

Format 1:

data-name-1 [(subscript-1[, subscript-2[, subscript-3]] ( ]

Format 2:

data-name-1 ({ index-name-1 | literal-1 } [{ + | - } literal-2]
[, { index-name-2 | literal-3 } [{ + | - } literal-4] [, { index-name-3 | literal-5 } [{ + | - } literal-6]]]

Restrictions on subscripting and indexing are:

1. A data-name must not itself be subscripted nor indexed when that data-name is being used as an index, or subscript.

2. Indexing is not permitted where subscripting is not permitted.

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.

PROGRAM STRUCTURE

A CIS 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.

THE "ANSI SWITCH" COMPILER DIRECTIVE

Some of the 'red-tape' statements required by a strict ANSI interpretation, are optional under CIS COBOL. It is possible to force the compiler to insist on a strict ANSI interpretation by using the "FLAG" directive. In the remainder of this Chapter these statements are marked [ ]. Elsewhere in this manual a reference is made to the ANSI switch when this applies.

If the operator enters the FLAG directive at compile time ANSI requirements implemented in CIS COBOL are mandatory depending on their level as specified by the Federal Compiler Testing Center under the direction of the General Services Administration (GSA). See the description of the Compiler FLAG directive in the *CIS COBOL Operating Guide*.

IDENTIFICATION DIVISION

GENERAL DESCRIPTION

The Identification Division must be included in every ANSI COBOL source program, This division identifies both the source program and the resultant output listing. In addition, the user may include the date the program is written, the date the compilation of the source program is accomplished and such other information as desired under the paragraphs in the general format shown below.

ORGANISATION

Paragraph headers identify the type of information contained in the paragraph. The name of the program must be given in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional and may be included in this division at the user's choice, in order of presentation shown by the format below.

STRUCTURE

The following is the general format of the paragraphs in the Identification Division and it defines the order of presentation in the source program.

General format

[IDENTIFICATION DIVISION.]
[PROGRAM-ID. program-name.]
[AUTHOR. [comment-entry]...]
[INSTALLATION. [comment-entry]...]
[DATE-WRITTEN. [comment-entry]...]
[DATE-COMPILED. [comment-entry]...]
[SECURITY. [comment-entry]...]

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.

General Format

```
┤ENVIRONMENT DIVISION.├
┤CONFIGURATION SECTION.├
┤SOURCE-COMPUTER. source-computer-entry├
┤OBJECT-COMPUTER. object-computer-entry├
[SPECIAL-NAMES. special-names-entry]
┤INPUT-OUTPUT SECTION.├
┤FILE-CONTROL.├ {file-control-entry}...
[I-O-CONTROL. input-output-control-entry]
```

DATA DIVISION

OVERALL APPROACH

The Data Division describes the data that the object program is to accept as input, to manipulate, to create, or to produce as output, Data to be processed falls into three categories:

1. That which is contained in files and enters or leaves the internal memory of the computer from a specified area or areas.
2. That which is developed internally and placed into intermediate or working storage, or placed into specific format for output reporting purposes.

3. Constants which are defined by the user.

PHYSICAL AND LOGICAL ASPECTS OF DATA DESCRIPTION

Data Division Organization

The DATA DIVISION which is one of the required divisions in a program, is subdivided into sections. These are the File, Working-Storage and Linkage sections.

The FILE SECTION defines the structure of data files. Each file is defined by a file description entry and one or more record descriptions, or by a file description entry and one or more report description entries. 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.

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]...]...]
[WORKING-STORAGE SECTION.
[{ 77-level-description-entry | record-description-entry }]...]
[LINKAGE-SECTION.
[{ 77-level-description-entry | record-description-entry }]...]

PROCEDURE DIVISION

GENERAL DESCRIPTION

The Procedure Division must be included in every COBOL source program. This division may contain declarative procedures.

Declaratives

Declarative sections must be grouped at the beginning of the Procedure Division preceded by the key word DECLARATIVES and followed by the key words END DECLARATIVES. (See descriptions of the USE statement in Chapters 5, 6 and 7 and the Debug Chapter 10).

Procedures

A procedure is composed of a paragraph, or group of successive paragraphs (the first paragraph-name is optional), or a section, or a group of successive sections within the Procedure Division. If one paragraph is in a section, then all paragraphs must be in sections. A procedure-name is a word used to refer to a paragraph or section in the source program in which it occurs. It consists of a paragraph-name (which may be qualified), or a section-name.

The end of the Procedure Division and the physical end of the program is that physical position in a COBOL source program after which no further procedures appear.

A section consists of a section header followed by zero, one, or more successive paragraphs. A section ends immediately before the next section or at the end of the Procedure Division or, in the declaratives portion of the Procedure Division, at the key words END DECLARATIVES.

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.

In CIS COBOL paragraph names may be entirely omitted. If paragraph names are used then they may be mixed with section names as required, in any order. Note that is is not possible to GO TO or PERFORM a piece of code unless it has either a section or a paragraph name.

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]...] .

Procedure Division Body

The body of the Procedure Division must conform to one of the following formats:

Format 1:

[DECLARATIVES. { section-name SECTION [segment-number] . declarative-sentence
[paragraph-name. [sentence]...]... }...
END DECLARATIVES.]
{ [section-name SECTION [segment-number]]
[[paragraph-name] [sentence]...] }

Format 2 :

{ [paragraph-name] [sentence]... }...

STATEMENTS AND SENTENCES

There are three types of statements:

1. Conditional statements,
2. Compiler directing statements,
3. Imperative statements.

There are three types of sentences:

1. Conditional sentences,
2. Compiler directing sentences,
3. Imperative sentences.

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 statement.
- A READ statement that specifies the AT END or INVALID KEY phrase.
- A WRITE statement that specifies the INVALID KEY phrase.
- A START, REWRITE or DELETE statement that specifies the INVALID KEY phrase.
- An arithmetic statement (ADD, DIVIDE, MULTIPLY, SUBTRACT) that specifies the SIZE ERROR phrase.
- A 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 9, 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.

Compiler Directing Sentence

A compiler directing sentence is a single compiler directing statement terminated by a period followed by a space.

Imperative Statement

An Imperative statement indicates a specific unconditional action to be taken by the object program. An imperative statement is any statement that is neither a conditional statement, nor a compiler directing statement. An imperative statement may consist of a sequence of imperative statements, each possibly separated from the next by a separator.

The imperative verbs are:

ACCEPT	DIVIDE[1]	READ[4]
ADD[1]	EXIT	REWRITE[2]
ALTER	GO	SET
CALL[3]	INSPECT	START[2]
CANCEL	MOVE	STOP
CLOSE	MULTIPLY[1]	SUBTRACT[1]
DELETE[2]	OPEN	WRITE[5]
DISPLAY	PERFORM	

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 AT END phrase or INVALID KEY phrase.
5. 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.

REFERENCE FORMAT

GENERAL DESCRIPTION

The reference format, which provides a standard method for describing COBOL source programs, is described in terms of character positions in a line on an input-output medium. The CIS COBOL compiler accepts source programs written in reference format and produces an output listing of the source program input in reference format.

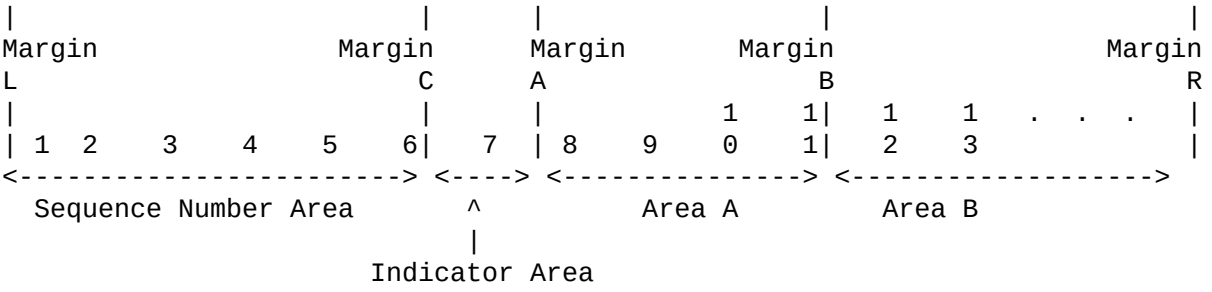
The rules for spacing given in the discussion of the reference format take precedence over all other rules for spacing.

The divisions of a source program must be ordered as follows: the Identification Division, then the Environment Division, then the Data Division, then the Procedure Division. Each division must be written according to the rules for the reference format.

REFERENCE FORMAT REPRESENTATION

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

Figure 2.1. Reference Format for a COBOL Source Line.



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.

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 8 and terminates immediately to the left of Margin R.

Sequence Numbers

A sequence number, consisting of six digits in the sequence area, may be used to label a source program line.

Continuation of Lines

Whenever a sentence, entry, phrase, or clause requires more than one line, it may be continued by starting subsequent line(s) in area B. These subsequent lines are called the continuation line(s). The line being continued is called the continued line. Any word or literal may be broken in such a way that part of it appears on a continuation line.

A hyphen in the indicator area of a line indicates that the first nonblank character in area B of the current line is the successor of the last nonblank character of the preceding line without any intervening space. However, if the continued line contains a nonnumeric literal without closing quotation mark, the first nonblank character in area B on the continuation line must be a quotation mark, and the continuation starts with the character immediately after that quotation mark. All spaces at the end of the continued line are considered part of the literal. Area A of a continuation line must be blank.

If there is no hyphen in the indicator area of a line, it is assumed that the last character in the preceding line is followed by a space.

Blank Lines

A blank line is one that is blank from margin C to margin R, inclusive. A blank line can appear anywhere in the source program, except immediately preceding a continuation line. (See Figure 2-1).

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.

Note that in CIS COBOL program sentences may commence anywhere in Area A or Area B.

When the sentences or entries of a paragraph require more than one line they may be continued as described in CONTINUATION OF LINES in this Chapter.

DATA DIVISION ENTRIES

Each Data Division entry begins with a level indicator or a level-number, followed by a space, followed by its associated name (except in the Report Section), followed by a sequence of independent descriptive clauses. Each clause, except the last clause of an entry, may be terminated by either the separator semicolon or the separator comma. The last clause is always terminated by a period followed by a space.

There are two types of Data Division entries: those which begin with a level indicator and those which begin with a level-number.

A level indicator is the indicator: FD (see THE FILE DESCRIPTION - COMPLETE ENTRY SKELETON in Chapters 5, 6 and 7)

In those Data Division entries that begin with a level indicator, the level indicator begins in area A followed by a space and followed in area B with its associated name and appropriate descriptive information.

Those Data Division entries that begin with level-numbers are called data description entries.

A level-number has a value taken from the set of values 1 through 49, 77. 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. At least one space must separate a level-number from the word following the level-number.

In those data description entries that begin with level-number 01 or 77, the level-number begins in area A followed by a space and followed in area B by its associated record-name or item-name and appropriate descriptive information.

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 tile 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 Appendix A.

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.

IDENTIFICATION DIVISION IN THE NUCLEUS

GENERAL DESCRIPTION

The Identification Division must be included in every COBOL source program. This division identifies the source program and the resultant output listing. In addition, the user may include the date the program is written and such other information as desired under the paragraphs in the general format shown below.

ORGANIZATION

Paragraph headers identify the type of information contained in the paragraph. The name of the program must be given in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional and may be included in this division at the user's choice, in the order of presentation shown by the general format below.

Structure

The general format of the paragraphs in the Identification Division is given below and shows the order of presentation in the source program.

General Format

```
┤IDENTIFICATION DIVISION├
┤PROGRAM-ID. program-name.├
[AUTHOR. [comment-entry]...]
[INSTALLATION. [comment-entry]...]
[DATE-WRITTEN. [comment-entry]...]
[DATE-COMPILED. [comment-entry]...]
[SECURITY. [comment-entry]...]
```

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 if the FLAG directive is used.
2. The program-name identifies the source program and all listings pertaining to a particular program.

THE DATE-COMPILED PARAGRAPH

Function

The DATE-COMPILED paragraph provides the compilation date in the Identification Division source program listing.

General Format

```
DATE-COMPILED. comment-entry ...
```

Syntax Rule

The comment-entry may be any combination of the characters from the computer's character set. The continuation of the comment entry by use of the hyphen is not permitted; however, the comment entry may be contained on one or more lines.

General Rule

The paragraph-name DATE-COMPILED causes a date string to be inserted during program compilation. If a DATE-COMPILED is present, the comment-entry is replaced in its entirety by the date string. See the *CIS COBOL Operating Guide* for details of the derivation of the comment-entry replacement string for your implementation of CIS COBOL compile-time.

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 treated as for documentation purposes only.

The OBJECT-COMPUTER Paragraph

Function

The OBJECT-COMPUTER Paragraph identifies the computer on which the program is to be executed.

General Format

```
OBJECT-COMPUTER. computer-name.
[,MEMORY SIZE integer { WORDS | CHARACTERS | MODULES } ]
[,PROGRAM COLLATING SEQUENCE IS alphabet-name]
[,SEGMENT-LIMIT IS segment-number]
```

Syntax Rules

1. Computer-name must be one COBOL word defined by the user.
2. Segment-number must be an integer in the range 1 through 49.

General Rules

1. The computer-name provides a means for identifying equipment configuration, in which case the computer-name and its implied configurations are specified by the user. The configuration definition contains specific information concerning the memory size. The computer-name, segment-limit and configuration definition are treated as for documentation purposes only.
2. If the PROGRAM COLLATING SEQUENCE Clause is specified, the collating sequence associated with alphabet-name is used to determine the truth value of any nonnumeric comparisons:

   Explicitly specified in relation conditions (see Relation Condition later in this Chapter).
3. If the PROGRAM COLLATING SEQUENCE Clause is not specified, the native collating sequence is used. Appendix B 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.

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.

General Format

```
SPECIAL-NAMES.
SWITCH {0 ... 7} [IS mnemonic-name] { ,ON STATUS IS condition-name-1 | [,OFF STATUS IS condition-name-2] | ,OFF STATUS IS condition-name-2 | [,ON STATUS IS condition-name-1]}
[ { ,SYSIN | ,SYSOUT } IS mnemonic-name ]
[, TAB IS mnemonic-name]
```

[, alphabet-name IS { STANDARD-1 | NATIVE }]...
[, CURRENCY SIGN IS literal-9]
[, DECIMAL-POINT IS COMMA]
[, CONSOLE IS CRT]
[, CURSOR IS data-name-1] .

General Rules

1. The status of the switch is specified by condition-names and interrogated by testing the condition-names (see Switch-Status Condition later in this Chapter).

2. The alphabet-name clause provides a means for relating a name to a specified character code set and/or collating sequence. When alphabet - name is referenced in the PROGRAM COLLATING SEQUENCE clause (see THE OBJECT-COMPUTER PARAGRAPH in this Chapter). The alphabet-name clause specifies a collating sequence. When alphabet-name is referenced in a CODE-SET clause in a file description entry (see The File Description Complete Entry Skeleton in Chapter 5), the alphabet-name clause specifies a character code set.

   a. If the STANDARD-1 phrase is specified, the character code set or collating sequence identified is that defined in American National Standard Code for Information Interchange, X3.4-1968 . Appendix B defines the correspondence between the characters of the standard character set and the characters of the native character set.

   b. If the NATIVE phrase is specified, the native character code set or native collating sequence is used. The native collating sequence is as in ANSI publication X3.4-1968 (see Appendix B) .

3. The character that has the highest ordinal position in the program collating sequence specified is associated with the figurative constant HIGH- VALUE. If more than one character has the highest position in the program collating sequence, the last character specified.

4. The character that has the lowest ordinal position in the program collating sequence specified is associated with the figurative constant LOW-VALUE. If more than one character has the lowest position in the program collating sequence, the first character specified is associated with the figurative constant LOW-VALUE.

5. The literal which appears in the CURRENCY SIGN IS literal clause is used in the PICTURE clause to represent the currency symbol. The literal is limited to a single character and must not be one of the following characters.

   - digits 0 thru 9;
   - alphabetic characters A, B, C, D, L, P, R, S, V, X, Z, or the space;
   - special characters '*', '+', '-', ',', '.', ';', '(', ')', '"', '/' or '='.

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

6. 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.

7. The clause CONSOLE IS changes the defaults in the ACCEPT and DISPLAY statements to the CIS 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.

8. The clause CURSOR IS 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 clause enables a program to retain a 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 of the 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. If data-name is zero, the effect is as if the CURSOR clause was not used, i.e., initial cursor position is top left of the screen. (See also the ACCEPT STATEMENT later in this Chapter).

9. SYSIN and SYSOUT specify the system input stream and system output stream respectively. At this release they are treated as for documentation purposes only.

10. TAB specifies the skip-to-head-of-form system function that can be used with WRITE ADVANCING. It is treated as for documentation purposes only at this release.

DATA DIVISION IN THE NUCLEUS

WORKING STORAGE SECTION

The Working-Storage Section is composed of the section header, followed by data description entries for noncontiguous data items and/or record description entries. Each Working-Storage Section record name and noncontiguous item name must be unique.

Noncontiguous Working-Storage

Items and constants in Working-Storage which bear no hierarchical relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Instead, they are classified and defined in a separate data description entry which begins with the special level-number, 77.

The following data clauses are required in each data description entry:

- Level-number 77
- Data-name
- The PICTURE clause or the USAGE IS INDEX clause

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.

General Format

level-number { data-name-1 | FILLER }
[; REDEFINES data-name-2]
[{ PICTURE | PIC } IS character-string]
[; [USAGE IS] { COMPUTATIONAL | COMP | COMPUTATIONAL-3 | COMP-3 | DISPLAY }]
[; [SIGN IS] { LEADING | TRAILING } [SEPARATE CHARACTER]]
[; { SYNCHRONIZED | SYNC } { LEFT | RIGHT }]
[; { JUSTIFIED | JUST } RIGHT] [; BLANK WHEN ZERO]
[; VALUE IS literal]

Syntax Rules

1. The level-number 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.

General Rule

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

THE BLANK WHEN ZERO CLAUSE

Function

The BLANK WHEN ZERO clause permits the blanking of an item when its value is zero.

General Format

```
BLANK WHEN ZERO
```

Syntax Rules

The BLANK WHEN ZERO clause can be used only for an elementary item whose PICTURE is specified as numeric with implicit or explicit USAGE IS DISPLAY, or numeric edited. (See the section called “THE PICTURE CLAUSE” later in this Chapter).

General Rules

1. When the BLANK WHEN ZERO clause is used, the item will contain nothing but spaces if the value of the item is zero.

2. When the BLANK WHEN ZERO clause is used for an item whose PICTURE is numeric, the category of the item is considered to be numeric edited.

THE DATA-NAME OR FILLER CLAUSE

Function

A data-name specifies the name of the data being described. The word FILLER specifies an elementary item of the logical record that cannot be referred to explicitly.

General Format

{ data-name | FILLER }

Syntax Rule

1. 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

1. The key word FILLER may be used to name an elementary item in a record. Under no circumstances can a FILLER item be referred to explicitly.

THE JUSTIFIED CLAUSE

Function

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

General Format

{ JUSTIFIED | JUST } RIGHT

Syntax Rules

1. The JUSTIFIED clause can be specified only at the elementary item level.

2. JUST is an abbreviation for JUSTIFIED.

3. The JUSTIFIED clause cannot be specified for any data item described as numeric or for which editing is specified.

General Rules

1. When a receiving data item is described with the JUSTIFIED clause and the sending data item is larger than the receiving data item, the leftmost characters are truncated. When the receiving data item is described with the JUSTIFIED clause and it is larger than the sending data item, the data is aligned at the rightmost character position in the data item with space fill for the leftmost character positions.

2. When the JUSTIFIED clause is omitted, the standard rules for aligning data within. an elementary item apply. (See Standard Alignment Rules.)

LEVEL NUMBER

Function

The level-number shows the hierarchy of data within a logical record. In addition, it is used to identify entries for working storage items, linkage items.

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 a File Description entry must have level-numbers with the values 01-49. (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.

General Rules

1. The level-number 01 identifies the first entry in each record description or a report group.
2. 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 section called “THE DATA DESCRIPTION - COMPLETE ENTRY SKELETON” in this Chapter).
3. Multiple level 01 entries subordinate to any given level indicator, represent implicit redefinitions of the same area.

THE PICTURE CLAUSE

Function

The PICTURE clause describes the general characteristics and editing requirements of an elementary item.

General Format

{ PICTURE | PIC } IS character-string

Syntax Rules

1. A PICTURE clause can be specified only at the elementary item level.
2. A character-string consists of certain allowable combinations of characters in the COBOL character set used as symbols. The allowable combinations determine the category of the elementary item.
3. The maximum number of characters allowed in the character-string is 30.
4. The PICTURE clause must be specified for every elementary item except an index data item, in which case use of this clause is prohibited.
5. PIC is an abbreviation for PICTURE
6. The asterisk when used as the zero suppression symbol and the clause BLANK WHEN ZERO may not appear in the same entry.

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) upper-case letters of the Roman alphabet and the space from the COBOL character set.

Numeric Data Rules

1. The 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. The 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. The contents when represented in standard data format can consist of any characters in the computer's character set.

Alphanumeric Edited Data Rules

1. Its PICTURE character-string is restricted to certain combinations of the following symbols: 'A', 'X', '9', 'B', '0', and'/' as follows:

   a. The character-string must contain at least one 'B' and at least one 'X' or at least one '0' (zero) and at least one 'X' or at least one '/' (stroke) and at least one 'X'; or

   b. The character-string must contain at least one '0' (zero) and at least one 'A' or at least one '/' (stroke) and at least one 'A'.

2. The contents when represented in standard data format are allowable characters in the computer's set.

Numeric Edited Data Rules

1. Its PICTURE character-string is restricted to certain combinations of the symbols 'B', '/', 'P', 'V', 'Z', '0', '9', ',', '.', '*', '+', '-', 'CR', 'DB', and the currency symbol. The allowable combinations are determined from the order of precedence of symbols and the editing rules as follows:

   a. The number of digit positions that can be represented in the PICTURE character-string must range from 1 to 18 inclusive.

   b. The character-string must contain at least one '0', 'B', '/', 'Z', '*', '+', ',', '.', '-', 'CR', 'DB', or currency symbol.

2. The contents of the character positions of these symbols that are allowed to represent a digit in standard data format, must be one of the numerals.

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.

S - The letter 'S' is used in a character-string to indicate the presence, but neither the representation nor, necessarily, the position of an operational sign; it must be written as the leftmost character in the PICTURE. The S is not counted in determining the size (in terms of standard data format characters) of the elementary item unless the entry is subject to a SIGN clause which specifies the optional SEPARATE CHARACTER phrase. (See the SIGN Clause in this Chapter.)

V - The 'V' is used in a character-string to indicate the location of the assumed decimal point and may only appear once in a character-string. The 'V' does not represent a character position and therefore is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string the 'V' is redundant.

X - Each 'X' in the character-string is used to represent a character position which contains any allowable character from the computer's character set.

Z - Each 'Z' in a character-string may only be used to represent the leftmost numeric character positions which will be replaced by a space character when the contents of that character position is zero. Each 'Z' is counted in the size of the item.

9 - Each '9' in the character-string represents a character position which contains a numeral and is counted in the size of the item.

0 - Each '0' (zero) in the character-string represents a character position into which the numeral zero will be inserted. The '0' is counted in the size of the item.

/ - Each '/' (stroke) in the character-string represents a character position into which the stroke character will be inserted. The '/' is counted in the size of the item.

, - Each ',' (comma) in the character-string represents a character position into which the character ',' will be inserted. This character position is counted in

the size of the item. The insertion character ',' must not be the last character in the PICTURE character-string.

. - The character '.' (period) in the character-string 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

There are two types of suppression and replacement editing:

- Zero suppression and replacement with spaces
- Zero suppression and replacement with asterisks

The type of editing which may be performed upon an item is dependent upon the category to which the item belongs. Table 3-1 specifies which type of editing may be performed upon a given category.

Table 3.1. Editing Types for Data Categories

CATEGORY	TYPE OF EDITING
Alphabetic	Simple insertion 'B' only
Numeric	None
Alphanumeric	None
Alphanumeric Edited	Simple insertion '0', 'B' and '/'

CATEGORY	TYPE OF EDITING
Numeric Edited	All, but see NOTE below

Note

Floating insertion editing and editing by zero suppression and replacement are mutually exclusive in a PICTURE clause. Only one type of replacement may be used with zero suppression in a PICTURE clause.

Simple Insertion Editing

Simple Insertion Editing. The ',' (comma), 'B' (space), '0' (zero), and '/' (stroke) are used as the insertion characters. The insertion characters are counted in the size of the item and represent the position in the item into which the character will be inserted.

Special Insertion Editing

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

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

Floating Insertion Editing

The currency symbol and editing sign control symbols '+' or '-' are the floating insertion characters and as such are mutually exclusive in a given PICTURE character-string.

Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two of the floating insertion characters. This string of floating insertion characters may contain any of the fixed insertion symbols or have fixed insertion characters immediately to the right of this string. These simple insertion characters are part of the floating string.

The leftmost character of the floating insertion string represents the leftmost limit of the floating symbol in the data item. The rightmost character of the floating string represents the rightmost limit of the floating symbols in the data item.

The second floating character from the left represents the leftmost limit of the numeric data that can be stored in the data item. Non-zero numeric data may replace all the characters at or to the right of this limit.

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

If the insertion characters are only to the left of the decimal point in the PICTURE character-string, the result is that a single floating insertion character will be placed into the character position immediately preceding either the decimal point or the first non-zero digit in the data represented by the insertion symbol string, whichever is farther to the left in the PICTURE character-string. The character positions preceding the insertion character are replaced with spaces.

If all numeric character positions in the PICTURE character-string are represented by the insertion character, the result depends upon the value of the data. If the value is zero the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point.

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of non-floating insertion characters being edited into the receiving data item, plus one for the floating insertion character.

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.

Precedence Rules

Table 3-3 shows the order of precedence when using characters as symbols in a character-string. An 'X' at an intersection indicates that the symbol(s) at the top of the column may precede, in a given character-string, the symbol(s) at the left of the row. Arguments appearing in braces indicate that the symbols are mutually exclusive. The currency symbol is indicated by the symbol 'cs'.

At least one of the symbols 'A', 'X', 'Z', '9' or '*', or at least two of the symbols '+', '-' or 'cs' must be present in a PICTURE string.

Table 3.3. PICTURE Character Precedence Chart.

First symbol	**Nonfloating Insertion Symbols**									**Floating Insertion Symbols**						**Other Symbols**					
Second symbol	**B**	**0**	**/**	**.**	**,**	**{+ -}**	**{+ -}**	**{CR DB}**	**cs**	**{Z *}**	**{Z *}**	**{±}**	**{±}**	**cs**	**cs**	**9**	**A X**	**S**	**V**	**P**	**P**
Nonfloating 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					
{+ -}[1]																					
{+ -}[2]	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 *}[1]	x	x	x	x		x			x	x											
{Z *}[2]	x	x	x	x	x	x			x	x	x								x		x
{+ -}[1]	x	x	x	x					x				x								
{+ -}[2]	x	x	x	x	x				x			x	x						x		x
cs[1]	x	x	x	x		x								x							
cs[2]	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[1]	x	x	x	x		x			x	x		x		x		x		x		x	
P[2]						x			x									x	x		x

In Table 3-3, non-floating insertion symbols '+' and '-', floating insertion symbols 'Z', '*' '+', '-', and 'cs', and other symbol 'P' appear twice in the PICTURE character precedence chart. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of symbol in the row and column represents its use to the right of the decimal point position.

THE REDEFINES CLAUSE

Function

The REDEFINES clause allows the same computer storage area to be described by different data description entries.

General Format

level-number *data-name-1*; REDEFINES *data-name-2*

Note

Level-number, data-name-1 are shown in the above format to improve clarity. Level-number and data-name-1 are not part of the REDEFINES clause.

Syntax Rules

1. The REDEFINES clause, when specified, must immediately follow data-name-1.
2. The level-numbers of data-name-1 and data-name-2 must be identical.
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. The data description entry for data-name-2 cannot contain an OCCURS clause. 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. 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.
5. Multiple level 01 entries subordinate to any given level indicator represent implicit redefinitions of the same area.

THE SIGN CLAUSE

Function

The SIGN clause specifies the position and the mode of representation of the operational sign when it is necessary to describe these properties explicitly.

General Format

[SIGN IS] { LEADING | TRAILING } [SEPARATE CHARACTER]

Syntax Rules

1. The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character 'S', or a group item containing at least one such numeric data description entry.

2. The numeric data description entries to which the SIGN clause applies must be described as USAGE IS DISPLAY.

3. At most one SIGN clause may apply to any given numeric data description entry.

4. If the CODE-SET clause is specified, any signed numeric data description entries associated with that file description entry must be described with the SIGN IS SEPARATE clause.

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. Th, 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, the section called “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.

THE SYNCHRONIZED CLAUSE

Function

The SYNCHRONIZED clause specifies the alignment of an elementary item on the natural boundaries of the computer memory.

General Format

{ SYNCHRONIZED | SYNC } [{ LEFT | RIGHT }]

Syntax Rules

1. This clause may only appear with an elementary item.
2. SYNC is an abbreviation for SYNCHRONIZED.

General Rules

1. The SYNCHRONIZED clause is accepted for documentation purposes only.
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 i.n 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.
   b. Any implicit FILLER generated for other data items within that same table are generated tor each occurrence of those data items.

9. This clause is hardware dependent.

THE USAGE CLAUSE

Function

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

General Format

[USAGE IS] { COMPUTATIONAL | COMP | DISPLAY | COMPUTATIONAL-3 | COMP-3 }

Syntax Rules

1. The PICTURE character-string of a COMPUTATIONAL or COMPUTATIONAL-3 item can contain only '9's, the operational sign character 'S', the implied decimal point character 'V', one or more 'P's. (See the section called “THE PICTURE CLAUSE” earlier in this Chapter).

2. COMP is an abbreviation for COMPUTATIONAL.

General Rules

1. The USAGE clause can be written at any level. If the USAGE clause is written at 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.

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 us ed 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 the section called “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, the initial value of data items in the Communication Section.

General Format

VALUE is literal

Syntax Rules

1. The VALUE clause cannot be stated for any items whose size is variable. (See THE OCCURS CLAUSE in Chapter 4).

2. A signed numeric literal must have associated with it a signed numeric PICTURE character-string.

3. All numeric literal in a VALUE clause of an item must have a value which is within the range of values indicated by the PICTURE clause, and must not have a value which would require truncation of nonzero digits. Nonnumeric literals in a VALUE clause of an item must not exceed the size indicated by the PICTURE clause.

General Rules

1. The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the item. The following rules apply:

   a. If the category of the item is numeric, all literals in the VALUE clause must be numeric. If the literal defines the value of a working storage item, the literal is aligned in the data item according to the standard alignment rules. (See Standard Alignment Rules in Chapter 2).

   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 section called “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.

Data Description Entries

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

1. The VALUE clause cannot be used in the File Section.

2. In the Working-Storage Section, the VALUE clause may be used to specify the initial value of a data item; in which case the clause causes the item to assume the specified value at the start of the object program. If the VALUE clause is not used in an item's description, the initial value is undefined.

3. The VALUE clause cannot be used in the Linkage Section.

4. The VALUE clause must not be stated in a data description entry that contains an OCCURS clause, or in an entry that is subordinate to an entry containing an OCCURS clause. (See THE OCCURS CLAUSE in Chapter 4).

5. The VALUE clause must not be stated in a data description entry that contains a REDEFINES clause, or in an entry that is subordinate to an entry containing a REDEFINES clause.

6. If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal, and the group area is initialized without consideration for the individual elementary or group items contained within this group. The VALUE clause cannot be stated at the subordinate levels within this group.

7. The VALUE clause must not be written for a group containing items with descriptions, including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).

PROCEDURE DIVISION IN THE NUCLEUS

CONDITIONAL EXPRESSIONS

Conditional expressions identify conditions that art, tested to enable the object program to select between alternate paths of control depending upon the truth value of the condition. Conditional expressions are specified in the IF and PERFORM statements. There are two categories of conditions associated with conditional expressions: simple conditions and relation conditions. Each may be enclosed within any number of paired parentheses, in which case its category is not changed.

Simple Conditions

The simple conditions are the relation, class, switch-status, conditions. A simple condition has a truth value of 'true' or 'false'. The inclusion in parentheses of simple conditions does not change the simple truth value.

Relation Condition

A relation condition causes a comparison of two operands, each of which may be the data item referenced by an identifier, a literal. A relation condition has a truth value of 'true' if the relation exists between the operands. Comparison of two numeric operands is permitted regardless of the formats specified in their respective USAGE clauses. However, for all other comparisons the operands must have the same usage. If either of the operands is a group item, the nonnumeric comparison rules apply.

The general format of a relation condition is as follows:

{ identifier-1 | literal-1 } { IS [NOT] GREATER THAN | IS [NOT] LESS THAN | IS [NOT] EQUAL TO | IS [NOT] > | IS [NOT] < | IS [NOT] = } { identifier-2 | literal-2 }

Note

The required relational characters '<', '>' and '=' are not underlined to avoid confusion with other symbols such as '#' (Greater than or equal to)

The first operand (identifier-1 or literal-1) is called the subject of the condition; the second operand (identifier-2 or literal-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; e.g., '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 3-4.

Table 3.4. 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 confusion 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, in terms of number of digits represented, is not significant. Zero is considered a unique value regardless of the sign. Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered positive for purposes of comparison.

Comparison of Nonnumeric Operands:

For nonnumeric operands, or one numeric and one nonnumeric operand, a comparison is made with respect to a specified collating sequence of characters (see The OBJECT-COMPUTER Paragraph in this Chapter). If one of the operands is specified as numeric, it must be an integer data item or an integer literal and:

1. If the nonnumeric operand is an elementary data item or a nonnumeric literal, the numeric operand is treated as though it were moved to an elementary alphanumeric data item of the same size as the numeric data item (in terms of standard data format characters), and the contents of this alphanumeric data item were then compared to the nonnumeric operand. (See THE MOVE STATEMENT in this Chapter, and the PICTURE Character 'P' under the heading Symbols Used earlier in this Chapter).

2. If the numeric operand is a group item, the numeric operand is treated as though it were moved to a group item of the same size as the numeric data item (in terms of standard data format characters), and the contents of this group item were then compared to the nonnumeric operand. (See THE MOVE STATEMENT in this Chapter, and the PICTURE character 'P' under the Heading Symbols Used earlier in this Chapter).

3. A non-integer numeric operand cannot be compared to a nonnumeric operand.

The size of an operand is the total number of standard data format characters in the operand. Numeric and nonnumeric operands may be compared only when their usage is the same.

There are two cases to consider:

1. Operands of equal size - If the operands are of equal size, comparison effectively proceeds by comparing characters in corresponding character positions starting from the high order end and continuing until either a pair of unequal characters is encountered or the low order end of the operand is reached, whichever comes first. The operands are determined to be equal if all pairs of characters compare equally through the last pair, when the low order end is reached.

   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 Condition

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; e.g. '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' thru 'Z' and the space.

Switch-Status Condition

A switch-status condition determines the 'on' or 'off' status of an implementor-defined switch. The implementor-name and the 'on' or 'off' value associated with the condition must be named in the SPECIAL-NAMES paragraph of the Environment Division. The general format for the switch-status condition is as follows:

condition-name

The result of the test is true if the switch is set to the specified position corresponding to the condition-name.

COMMON PHRASES AND GENERAL RULES FOR STATEMENT FORMATS

In the statement descriptions that follow, several phrases appear frequently: the ROUNDED phrase, the SIZE ERROR phrase.

These are described below. A resultant-identifier is that identifier associated with a result of an arithmetic operation.

The Rounded Phrase

If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested the absolute value of the resultant-identifier is increased by one whenever the most significant digit of the the excess is greater than or equal to five.

When the low-order integer positions in a resultant-identifier are represented by the character 'P' in the PICTURE for the resultant-identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

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. After completion of the execution of this operation, the imperative statement in the SIZE ERROR phrase is executed.

Arithmetic Statements

The arithmetic statements are the ADD, DIVIDE, MULTIPLY, and SUBTRACT statements. Common features are as follows:

1. The data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment are supplied throughout the calculation.

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 section called “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, statement share a part of their storage areas, the result of the execution of such a statement is undefined.

Incompatible Data

Except for the class condition (See Class Condition in this Chapter), when the contents of a data item are referenced in the Procedure Division and the contents of that data item are not compatible with the class specified for that data item by its PICTURE clause, then the result of such a reference is undefined. If a numeric display field contains one or more spaces the spaces are usually treated as if they were zero. This may present problems in portability if relied upon.

CRT Devices

The CRT is driven directly by the run time system via a buffer. The COBOL programmer moves data into and out of this buffer by means of ACCEPT and DISPLAY statements. Each ACCEPT or DISPLAY action is relative to the start of the CRT buffer unless POSITION is specified. The syntax is limited to inputting to or outputting from a single data name. The data name may be a group item and several such group items may redefine the same area of storage.

The use of FILLER data items in record descriptions used for input or output to a CRT device is subject to special rules. On output, any FILLER item in a record results in suppression of output for the character positions it defines. On input, any FILLER item suppresses operator keying into the character positions it defines.

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 CONSOLE]
```

Format 2

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

Syntax Rule

Literal-1 must be numeric.

General Rules

1. Format 1 is the standard ANSI ACCEPT statement

   Format 2 is the extended ACCEPT format. The two formats are distinguished by their FROM phrases and the default assumes FROM CONSOLE. The default can, however, be changed by specifying CONSOLE IS CRT in the SPECIAL-NAMES clause so that FROM CRT becomes the default. This changed default is not shown in the syntax above. Note: Specifying the AT phrase implies Format 2, even if FROM CRT is omitted.

Format 1

2. The ACCEPT statement reads one line of input data from the system console device. This input data replaces the contents of the data item named by the identifier.

3. The line of input is line-edited according to the operating system rules for line-editing (see Operating Systems User Guide). The line is terminated by pressing the CR (Carriage Return) key or by exceeding 120 characters in length.

4. If the input line is of the same size as the receiving data item, the transferred data is stored in the receiving data item.

5. If the input line is not of the same size as the receiving data item, then:

   a. If the size of the receiving data item exceeds the size of the input line, the transferred data is stored aligned to the left in the receiving data item and the data item is filled with trailing spaces.

   b. If the size of the transferred data exceeds 120 bytes, only the first 120 characters of the input line are stored in the receiving data item. The remaining characters of the input line 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 is 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. Numeric items are treated as two separate integer numeric fields and edited fields are treated as Alphanumeric fields except as described in rule 12.

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. The most significant 99 is taken as the line count in the range one to the maximum lines on the user screen. The least significant 99 is taken as a character position in the range one to the maximum positions allowed by the screen width of the user CRT.

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 items it occurs and one definition is thus automatically repeated for many fields.

11. Immediately upon execution of the ACCEPT statement the cursor is positioned to the CRT location corresponding to the left-most non-FILLER character position in data-name-1. Alternatively, when CURSOR is specified in the SPECIAL-NAMES paragraph, the cursor is positioned at the position held in the CURSOR data-name in the same format as the screen position is held in data-name-2. If the cursor data-name has the value SPACE or ZERO, the effect is as if the CURSOR was not specified; if a valid screen position is specified that is not within a non-FILLER item, the cursor is positioned at the nearest non-FILLER character position. CURSOR data-name holds the last cursor position at the end of execution of an ACCEPT statement.

12. If FROM CRT is not specified, the default is FROM CONSOLE (see rule 1 above).

13. As the operator keys characters, the cursor moves to the right one character position at a time in locations corresponding to data fields. The operator always keys into the current cursor position. At the end of a line the cursor moves down one line and to the leftmost non-FILLER character position.

14. If the data item is integer numeric, only numeric characters (0 - 9) will be accepted into that item. Keying the decimal point character (. or , as specified in the DECIMAL POINT phrase) when accepting a numeric item causes the item to be right justified and zero-filled from the left.

15. When the cursor location reaches a position corresponding to a FILLER item in a data-name, it immediately skips to the next non-FILLER character position, or if there is no such position remaining in the portion of the CRT specified by the data-name, it remains in its current position.

16. The operator can terminate input by pressing the CR (Carriage Return) key at which time control is passed to the next statement after ACCEPT. Before control is passed to the next statement the following takes place:

    a. The numeric value of each numeric-edited data-field is formed internally from only the keyed characters 0 to 9, +, -, . or , and then moved back to the numeric-edited field with the ANSI PICTURE editing applied. The field may thus be different to that shown on the CRT just before the Carriage Return was pressed.

    b. When CURSOR IS is specified in the SPECIAL-NAMES paragraph, the cursor position when the Carriage Return is pressed is returned in the data-name specified by the CURSOR IS clause, except when its value at the start of the ACCEPT function caused it the be treated as unspecified.

17. Before keying CR, the operator can reposition the cursor to overwrite data already keyed or to skip character positions by use of the character position keys shown in Table 3-5.

    NOTE: The actual key identification and functions shown in this table vary according to the CRT used and the way it is configured (See the *CIS COBOL Operating Guide*).

Table 3.5. Cursor Repositioning Keys

Key	Function
←	Backs up the cursor one position
↑	Backs up the cursor to the start of the non-FILLER field prior to the current cursor position.
↓	Moves the cursor on to the start of the next non-FILLER field in advance of the current cursor position.
→	Moves the cursor on one position without overwriting existing contents.
#	Moves the cursor back to the start of the first non-FILLER field in the CRT area corresponding to data-name-1.

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

ADD { identifier-1 | literal-1 } [, { identifier-2 | literal-2 }]... TO identifier-m [ROUNDED] [, identifier-n[ROUNDED]]... [; ON SIZE ERROR imperative-statement]

Format 2

ADD { identifier-1 | literal-1 } , { identifier-2 | literal-2 } [, { identifier-3 | literal-3 }]... GIVING identifier-m [ROUNDED] [, identifier-n [ROUNDED]] [; ON SIZE ERROR imperative-statement]

Syntax Rules

1. In Formats 1 and 2, each identifier must refer to an elementary numeric item, except that in Format 2 each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

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.

General Rules

1. See the section called “The Rounded Phrase”, The Size Error Phrase, The Arithmetic Statements, Overlapping Operands and the section called “Incompatible Data” in this Chapter.

2. If Format 1 is used, the values of the operands preceding the word TO are added together, then the sum is added to the current value of identifier-m storing the result immediately into identifier-m.

3. If Format 2 is used, the value of the operands preceding the word GIVING are added together, then the sum is stored as the new value of identifier-m, the resultant identifiers.

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

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

Syntax Rule

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

2. Procedure-name-2 is the name of a paragraph or section in the Procedure Division.

General Rule

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

THE DISPLAY STATEMENT

Function

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

General Formats

Format 1

```
DISPLAY { identifier-1 | literal-1 } [ , { identifier-2 | literal-2 } ]... [UPON CONSOLE]
```

Format 2

```
DISPLAY { data-name-1 | literal-3 } [ AT { data-name-2 | literal-4 } ] UPON { CRT | CRT-UNDER }
```

Syntax Rules

Format 1

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

2. If the literal is numeric, it must be an unsigned integer.

Format 2

3. Literal-3 must be alphanumeric. Literal-4 must be numeric.

4. data-name-1 may refer to a record, group or elementary item, but it must not be subscripted.

General Rules

1. Format 1 is the standard ANSI DISPLAY statement.

   Format 2 is the extended DISPLAY format.

   The two formats are distinguished by their UPON phrases and the default assumes UPON CONSOLE. The default can, however, be changed by specifying CONSOLE IS CRT in the SPECIAL-NAMES clause so the UPON CRT becomes the default. This changed default is not shown in the syntax above. Note: Specifying the AT phrase implies Format 2, even if the UPON phrase is omitted.

Format 1

2. The DISPLAY statement causes the contents of each operand to be transferred to the CRT in the order listed as one line of output data.

3. The size of the data transfer can be up to 132 bytes.

4. If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

5. If the CRT is capable of displaying data of the same size as the data item being output, the data item is transferred.

6. If the CRT is not capable of displaying data of the same size as the data item being transferred, one of the following applies.

   a. If the size of the data item being displayed exceeds the size of the data that the CRT is capable of receiving in a single transfer, the data beginning with the leftmost character is stored aligned to the left in the receiving CRT.

   b. If the size of the data item that the CRT is capable of receiving exceeds the size of the data being transferred, the transferred data is stored aligned to the left in the receiving CRT.

7. When a DISPLAY statement contains more than one operand, the size of the sending item is the sum of the sizes associated with the operands, and the values of the operands are transferred in the sequence in which the operands are encountered.

Format 2

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

9. 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.

10. Elementary data items within data-name-1 may be alphanumeric, integer numeric, numeric or edited.

11. AT data-name-2 or literal-4 defines the position on the screen of the leftmost character of the. data. Either form must refer to a PIC 9999 field. The most significant 99 is taken as a line count in the range one to the maximum number of lines on the user screen. The least significant 99 is taken as a character position in the range one to the maximum of characters per line on the user screen.

12. data-name-1 may refer to a record, group or elementary item, but it may not be subscripted. REDEFINES may be used, 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.

13. DISPLAY SPACE has the effect of clearing the screen at run time (i.e. filling the whole screen with spaces). DISPLAY " " (one space character), however, displays only one space character.

14. The CRT-UNDER phrase causes the elementary items moved to the CRT to be displayed with the underline feature present. This feature is dependent on the CRT hardware functions and is not available on all makes of CRT (see the *CIS COBOL Operating Guide*).

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 | literal-1 } INTO identifier-2 [ROUNDED]
[, identifier-3 [ROUNDED]]... [; ON SIZE ERROR imperative-statement]

Format 2

DIVIDE { identifier-1 | literal-1 } INTO { identifier-2 | literal-2 } GIVING identifier-3 [ROUNDED]
[, identifier-4 [ROUNDED]]... [; ON SIZE ERROR imperative-statement]

Format 3

DIVIDE { identifier-1 | literal-1 } BY { identifier-2 | literal-2 } GIVING identifier-3 [ROUNDED]
[, identifier-4 [ROUNDED]]... [; ON SIZE ERROR imperative-statement]

Syntax Rules

1. Each identifier must refer to an elementary numeric item, except that any identifier associated with the GIVING phrase must refer to either an elementary numeric item or an elementary numeric edited item.

2. Each literal must be a numeric literal.

General Rules

1. See the section called “The Rounded Phrase”, The Size Error Phrase, The Arithmetic Statements, Overlapping Operands and the section called “Incompatible Data” in this Chapter for a description of these functions.

2. When Format 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.

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.

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.

THE ENTER STATEMENT

Function

The ENTER statement provides a means of allowing the use of more than one language in the same program.

General Format

ENTER language-name [routine-name]

Syntax Rule

1. This statement is for documentation purposes only.

General Rule

1. Access to other languages can be achieved by means of CALL.

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 :o 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.

THE GO TO STATEMENT

Function

The GO TO statement causes control to be transferred from one part of the Procedure Division to another.

General Format

Format 1

`GO TO` {procedure-name-1}

Format-2

`GO TO` procedure-name-1 [, procedure-name-2]... , procedure-name-n DEPENDING ON identifier

Syntax Rules

1. Identifier is the name of a numeric elementary item described without any positions to the right of the assumed decimal point.
2. When a paragraph is referenced by an ALTER statement, that paragraph can consist only of a paragraph header followed by a Format 1 GO TO statement.
3. If a GO TO statement represented by Format 1 appears in a consecutive sequence of imperative statements within a sentence, it appears as the last statement in that sequence.

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. 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.

THE IF STATEMENT

Function

The IF statement causes a condition to be evaluated (see CONDITIONAL EXPRESSIONS in this Chapter). The subsequent action of the object program depends on whether the value of the condition is true or false.

General Format

`IF` condition; `[THEN]` { statement-1 | NEXT SENTENCE } { ; ELSE statement-2 | ; ELSE NEXT SENTENCE }

Syntax Rules

1. Statement-1 and statement-2 represent either an imperative statement or a conditional statement, and either may be followed by a conditional statement.

2. The ELSE NEXT SENTENCE phrase may be omitted if it immediately precedes the terminal period of the sentence.

General Rules

1. When an IF statement is executed, the following transfers of control occur:

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

   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.

THE INSPECT STATEMENT

Function

The INSPECT statement provides the ability to tally (Format 1), replace (Format 2), or tally and replace (Format 3) occurrences of single characters in a data item.

General Format

Format 1

```
INSPECT identifier-1 TALLYING identifier-2 FOR , { ALL | LEADING | CHARACTERS }
{ identifier-3 | literal-1 } [ { BEFORE | AFTER } INITIAL { identifier-7 | literal-5 } ]
```

Format 2

```
INSPECT identifier-1 REPLACING
{ CHARACTERS BY identifier-6 | literal-4 | , { ALL | LEADING | FIRST } , { identifier-5 | literal-3 }
BY { identifier-6 | literal-4 } }
[ { BEFORE | AFTER } INITIAL { identifier-7 | literal-5 } ]
```

Format 3

```
INSPECT identifier-1 TALLYING identifier-2 FOR , { ALL | LEADING | CHARACTERS }
{ identifier-3 | literal-1 } [ { BEFORE | AFTER } INITIAL { identifier-4 | literal-2 } ]
REPLACING
{ CHARACTERS BY identifier-6 | literal-4 | , { ALL | LEADING | FIRST } , { identifier-5 | literal-3 }
BY { identifier-6 | literal-4 } }
[ { BEFORE | AFTER } INITIAL { identifier-7 | literal-5 } ]
```

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. In Level 1, 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 must be one character in length.

Formats 1 and 3 Only

5. Identifier-2 must reference an elementary numeric data item.

6. If either literal-1 or literal-2 is a figurative constant, the figurative constant refers to an implicit one character data item.

Formats 2 and 3 Only

7. The size of the data referenced by literal-4 or identifier-6 must be equal to the size of the data referenced by literal-3 or identifier-5. When a figurative constant is used as literal-4, the size of the figurative constant is equal to the size of literal-3 or the size of the data item referenced by identifier-5.

8. When the CHARACTERS phrase is used, literal-4, literal-5, or the size of the data item referenced by identifier-6, identifier-7 must be one character in length.

9. When a figurative constant is used as literal-3, the data referenced by literal-4 or identifier-6 must be one character in length.

General Rules

All Formats

1. Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying and/or replacing) begins at the leftmost character position of the data item referenced by identifier-1, regardless of its class, and proceeds from left to right to the rightmost character position as described in general rules 4 through 6.

2. For use in the INSPECT statement, the contents of the data item referenced by identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 will be treated as follows:

   a. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 are described as alphanumeric, the INSPECT statement treats the contents of each such identifier as a character-string.

   b. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 are 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 the identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 are 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). Data items to be referenced by the INSPECT verb should be placed such that they Lie within the first 10,000 bytes of intermediate code.

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 no match occurs in the comparison of the first literal-1, literal-3, the comparison is repeated with each successive literal-1, literal-3, if any, until either a match is found or there ls no next successive literal-1, literal-3. When there is no next successive literal-1, literal-3, the character position in the data item referenced by identifier-1 immediately to the right of the leftmost character position considered in the last comparison cycle is considered as the leftmost character position, and the comparison cycle begins again with the first literal-1, literal-3.

   c. Whenever a match occurs, tallying and/or replacing takes place as described in general rules 8 through 10. The character position in the data item referenced by identifier-1 immediately to the right of the rightmost character position that participated in the match is now considered to be the leftmost character position of the data item referenced by identifier-1, and the comparison cycle starts again with the first literal-1, literal-3.

   d. The comparison operation continues until the rightmost character position of the data item referenced by identifier-1 has participated in a match or has been considered as the leftmost character position. When this occurs, inspection is terminated.

   e. If the CHARACTERS phrase is specified, an implied one character operand participates in the cycle described in paragraphs 5a through 5d above, except that no comparison to the contents of the data item referenced by identifier-1 takes place. This implied character is considered always to match the leftmost character of the contents of the data item referenced by identifier-1 participating in the current comparison cycle.

6. The comparison operation defined in general rule 5 is affected by the BEFORE and AFTER phrases as follows:

a. If the BEFORE or AFTER phrase is not specified, literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates in the comparison operation as described in general rule 5.

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 are not initialized by the execution of the INSPECT statement.

8. The rules for tallying are as follows:

   a. If the ALL phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one for each occurrence of literal-1 matched within the contents of the data item referenced by identifier-1.

   b. If the LEADING phrase is specified, the contents of the data item referenced by identifier-2 are incremented by one for each contiguous occurrence of literal-1 matched within the contents of the data item referenced by identifier-1, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.

   c. If the CHARACTERS phrase is specified, the contents of the data item referenced by identifier-2 are incremented by one for each character matched, in the sense of general rule 5e, within the contents of the data item referenced by identifier-1.

Format 2

9. The required words ALL, LEADING, and FIRST are adjectives.

10. The rules for replacement are as follows:

    a. When the CHARACTERS phrase is specified, each character matched in the sense of general rule 5e in the contents of the data item referenced by identifier-1 is replaced by literal-4.

b. When the adjective ALL is specified, each occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4.

c. When the adjective LEADING is specified, each contiguous occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4, provided that the leftmost occurrence is at the point where comparison began in the first comparison cycle in which literal-3 was eligible to participate.

d. When the adjective FIRST is specified, the leftmost occurrence of literal-3 matched within the contents of the data item referenced by identifier-1 is replaced by literal-4.

Format 3

1. -. 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

Four examples of the use of the INSPECT statement follow:

```
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"
```

Where word = ADJECTIVE, count = 6, word = BDJECTIVE.
Where word = JACK, count = 3, word = JBCK.
Where word = JUJMAB, count = 5, word = JUJMBB.

```
INSPECT word REPLACING CHARACTERS BY "B" BEFORE INITIAL "A".
```

word before: 1 2 X Z A B C D
word after: B B B B B A B C D

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 | literal } TO identifier-2 [, identifier-3 ...]
```

Syntax Rules

1. Identifier-1 and literal represent the sending area; identifier-2, identifier-3, ... represent the receiving area.
2. An index data item cannot appear as an operand of a MOVE statement. (See THE USAGE CLAUSE in this Chapter).

General Rules

1. The data designated by the literal or identifier-1 is moved first to identifier-2, then to identifier-3, ... The rules referring to identifier-2 also apply to the other receiving areas. Any subscripting or indexing associated with identifier-2, ..., is evaluated immediately before the data is moved to the respective data item.

   Any subscripting or indexing associated with identifier-1 is evaluated only once, immediately before data is moved to the first of the receiving operands. The result of the statement:

   ```
   MOVE a (b) TO b, c (b)
   ```

   is equivalent to:

   ```
   MOVE a (b) TO temp
   MOVE temp TO b
   MOVE temp TO c (b)
   ```

   where 'temp' is an intermediate result item provided by the implementor.

   See the section called “Incompatible Data” in this Chapter.

2. Any MOVE in which the sending and receiving items are both elementary items is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, alphanumeric edited. These categories are described in the PICTURE clause. Numeric literals belong to the category numeric, and nonnumeric literals belongs to the category alphanumeric. The figurative constant ZERO belongs to the category numeric. The figurative constant SPACE belongs to the category alphabetic. All other figurative constants belong to the category alphanumeric. The following rules apply to an elementary move between these categories:

   a. The figurative constant SPACE, alphanumeric edited, or alphabetic data item must not be moved to a numeric or 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. A move from a numeric edited field to a numeric field will work provided:

      i. the source field is not blank

ii. the source field does not contain non-stored editing characters i.e. P, S or V

iii. zero is not used as an edited character.

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

3. 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 zeroes are replaced because of editing requirements. When a signed numeric item is the receiving item, the sign of the sending item is placed in the receiving item. (See THE SIGN CLAUSE in this Chapter). Conversion of the representation of the sign takes place as necessary. If the sending item is unsigned, a positive sign is generated for the receiving item.

      When an unsigned numeric item is the receiving item, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.

      When a data item described as alphanumeric is the sending item, data is moved as if the sending item were described as an unsigned numeric integer.

   c. When a receiving field is described as alphabetic, justification and any necessary space-filling takes place as defined under the STANDARD ALIGNMENT RULES in Chapter 2. 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.

4. Any move that is not an elementary move is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area will be filled without consideration for the individual elementary or group items contained within either the sending or receiving area.

5. Data in Table 3-6 summarizes the legality of the various types of MOVE statements. The general rule reference indicates the rule that prohibits the move or the behavior of a legal move.

Table 3.6. MOVE Statement Data Categories.

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

Category of Sending Data Item	Category of Receiving Data Item[1]			
NUMERIC NON-INTEGER	No/2b	No/2c	Yes/3b	Yes/3b
NUMERIC EDITED	No/2b	Yes/3a	Yes/2d	No/2a
1 - The relevant rule number is quoted in these columns				

THE MULTIPLY STATEMENT

Function

The MULTIPLY statement causes numeric data items to be multiplied and sets the values of data items equal to the results.

General Format

Format 1

```
MULTIPLY { identifier-1 | literal-1 } BY identifier-2 [ROUNDED]
[, identifier-3 [ROUNDED]]... [; ON SIZE ERROR imperative-statement]
```

Format 2

```
MULTIPLY { identifier-1 | literal-1 } BY { identifier-2 | literal-2 } GIVING identifier-3 [ROUNDED]
[, identifier-4 [ROUNDED]]... [; ON SIZE ERROR imperative-statement]
```

Syntax Rules

1. Each identifier must refer to a numeric elementary item, except that in Format 2 each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

2. Each literal must be a numeric literal.

General Rules

1. See the section called “The Rounded Phrase”, The Size Error Phrase, The Arithmetic Statements, Overlapping Operands and the section called “Incompatible Data” 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.

THE PERFORM STATEMENT

Function

The PERFORM statement is used to transfer control explicitly to one or more procedures and to return control implicitly whenever execution of the specified procedure is complete.

General Format

Format 1

```
PERFORM procedure-name-1 [ { THROUGH | THRU } procedure-name-2 ]
```

Format 2

`PERFORM` procedure-name-1 [{ THROUGH | THRU } procedure-name-2] { identifier-1 | integer-1 } TIMES

Format 3

`PERFORM` procedure-name-1 [{ THROUGH | THRU } procedure-name-2] UNTIL condition-1

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. The words THRU and THROUGH are equivalent.

3. Where procedure-name-1 and procedure-name-2 are both specified and either is the name of a procedure in the declarative section of the program then both must be procedure-names in the same declarative section.

General Rules

1. When the PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1 (except as indicated in general rules 4b, 4c, and 4d). This transfer of control occurs only once for each execution of a PERFORM statement. For those cases where a transfer of control to the named procedure does take place, an implicit transfer of control to the next executable statement following the PERFORM statement is established as follows:

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

   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.

2. 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.

3. 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,

4. The PERFORM statements operate as follows with rule 3 above applying to all formats:

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

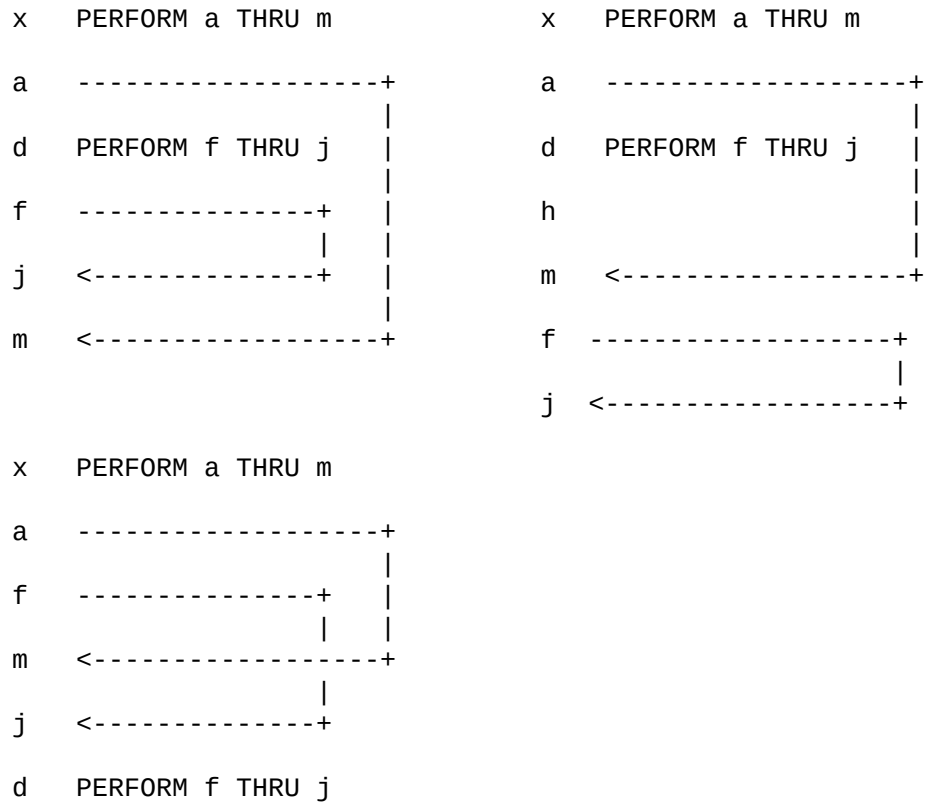
   b. Format 2 is the PERFORM TIMES. The procedures are performed the number of times specified by integer-1 or by the initial value of the data item referenced by identifier-1 for that execution. If, at the time of execution of a PERFORM statement, the value of the data item referenced

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. During execution of the PERFORM statement, references to identifier-1 cannot alter the number of times the procedures are to be executed from that which was indicated by the initial value of identifier-1.

c. Format 3 is the PERFORM...UNTIL. The specified procedures are performed until the condition specified by the UNTIL phrase is true. When the condition is true, control is transferred to the next executable statement after the PERFORM statement. If the condition is true when the PERFORM statement is entered, no transfer to procedure-name-1 takes place, and control is passed to the next executable statement following the PERFORM statement.

5. If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, must not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements may not have a common exit. See Figure 3-1.

Figure 3.1. PERFORM Statement in Sequence.



6. 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.

   c.

7. 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.

8. PERFORM statements must not be nested to greater than 22 levels.

THE STOP STATEMENT

Function

The STOP statement causes a permanent or temporary suspension of the execution of the object program.

General Format

```
STOP { RUN | literal }
```

Syntax Rules

1. The literal may be numeric or non-numeric or may be any figurative constant, except ALL.

2. If the literal is numeric, then it must be an unsigned integer.

3. If a STOP RUN statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

General Rules

1. If the RUN phrase is used, then the operating system ending procedure is instituted.

2. If STOP literal is specified, the literal is communicated to the operator. Continuation of the object program begins with the execution of the next executable statement in sequence.

THE SUBTRACT STATEMENT

Function

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from one or more items, and set the values of one or more items equal to the results.

General Format

Format 1

```
SUBTRACT { identifier-1 | literal-1 } , { identifier-2 | literal-2 }... ... FROM identifier-m [ROUNDED]
[ , identifier-n [ROUNDED] ]... [; ON SIZE ERROR imperative-statement]
```

Format 2

```
SUBTRACT { identifier-1 | literal-1 } , { identifier-2 | literal-2 }... ... FROM identifier-m GIVING
identifier-n [ROUNDED] [ , identifier-o [ROUNDED] ]... [; ON SIZE ERROR imperative-statement]
```

Syntax Rules

1. Each identifier must refer to a numeric elementary item except that in Format 2, each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

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.

General Rules

1. See the section called “The Rounded Phrase”, The Size Error Phrase, The Arithmetic Statement, Overlapping Operands and the section called “Incompatible Data” 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-n, etc.

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

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. The maximum number of multiple dimensions if the ANSI switch is on (see Chapter 2) is restricted to three.

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.

General Format

OCCURS integer-2 TIMES [INDEXED BY index-name-1 [, index-name-2]...]...

Syntax Rules

1. 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 since its allocation and format are dependent on the hardware, and not being data, cannot be associated with any data hierarchy.

2. The OCCURS clause cannot be specified in a data description entry that has 01 or 77 level-number (if ANSI switch has been set).

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

General Rules

1. The OCCURS clause is used in defining tables and other homogeneous sets of repeated data items. Whenever the OCCURS clause is used, the data-name which is the subject of this entry must be either subscripted or indexed whenever it is referred to in a statement other than USE FOR DEBUGGING. Further, if the subject of this entry is the name of a group item, then all data-names belonging to the group must be subscripted or indexed whenever they are used as operands, except as the object of a REDEFINES clause. (See under headings Subscripting, Indexing and Identifier in Chapter 2).

2. Except for the OCCURS clause itself, all data description clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described. (See restriction in general rule 2 under Data Description Entries Other Than Condition in Chapter 3).

3. The number of occurrences of the subject entry is defined as the value of integer-2 representing the exact number of occurrences.

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 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.

2. An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value which must correspond to an occurrence number of a table element. The elementary item cannot be a conditional variable. The compiler will allocate a 2 byte binary field with an implied picture of 9(4) COMPUTATIONAL. If a group item is described with the USAGE IS INDEX clause the elementary items in the group are all index data items. The group itself is not an index data item and cannot be used :n the SET statement or in a relation condition.

3. An index data item can be part of a group which is referred to in a MOVE or input-output statement, in which case no conversion will take place.

PROCEDURE DIVISION IN THE TABLE HANDLING MODULE

RELATION CONDITION

Comparisons Involving Index-Names And/or Index Data Items

Relation tests may be made between the following data items:

- Two index-names. The result is the same as if the corresponding occurrence numbers were compared.

- An index-name and a data item (other than an index data item) or literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal.

- An index data item and an index-name or another index data item. The actual values are compared without conversion.

- The result of the comparison of an index data item with any data item or literal not specified above is undefined.

OVERLAPPING OPERANDS

When a sending and a receiving item in a SET statement share a part of their storage areas, the result of the execution of such a statement is undefined.

THE SET STATEMENT

Function

The SET statement establishes reference points for table handling operations by setting index-names associated with table elements.

General Format

Format 1

SET { identifier-1 [, identifier-2]... | index-name-1 [, index-name-2]... } TO { identifier-3 | index-name-3 | integer-1 }

Format 2

SET index-name-4 [, index-name-5]... { UP BY | DOWN BY } { identifier-4 | integer-2 }

Syntax Rules

1. All references to index-name-1, identifier-1, and index-name-4 apply equally to index-name-2, identifier-2, and index-name-5, respectively.
2. Identifier-1 and identifier-3 must name either index data items, or elementary items described as an integer.
3. Identifier-4 must be described as an elementary numeric integer.
4. Integer-1 and integer-2 may be signed. Integer-1 must be positive.

General Rules

1. Index-names are considered related to a given table 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 PERFORM statement may be undefined. (See 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.

4. In Format 2, the contents of index-name-4 are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of integer-2 or identifier-4; thereafter, the process is repeated for index-name-5, etc. Each time the value of identifier-4 is used as it was at the beginning of the execution of the statement.

5. Data in Table 4-1 represents the validity of various operand combinations in the SET statement. The general rule reference indicates the applicable general rule.

Table 4.1. SET Statement Valid Operand Combinations.

Sending Item	Receiving Item[1]		
	Integer Data Item	Index-Name	Index Data Item
Integer Literal	No/3c	Valid/3a	No/3b
Integer Data Item	No/3c	Valid/3a	No/3b
Index-Name	Valid/3c	Valid/3a	Valid/3b[2]
Index Data Item	No/3c	Valid/3a[2]	Valid/3b[2]
1 = Rule numbers under General Rules above are referred to. 2 = No conversion takes place			

Chapter 5. SEQUENTIAL INPUT AND OUTPUT

INTRODUCTION TO THE SEQUENTIAL I-O MODULE

The Sequential I-O module provides a capability to access 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.

Current Record Pointer

The current record pointer is a conceptual entity used in this document to facilitate specification of the next record to be accessed within a given file. The concept of the current record pointer has no meaning for a file opened in the output mode. The setting of the current record pointer is affected only by the OPEN and READ statements.

I-O Status

If the FILE STATUS clause is specified in a file control entry, a value is placed into the specified two-character data item during the execution of an OPEN, CLOSE, READ, WRITE, or REWRITE statement and before any applicable USE procedure is executed, to indicate to the COBOL program the status of that input-output operation.

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' - indicates Successful Completion
'1' - indicates At End
'3' - indicates Permanent Error
'9' - indicates an 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 sequential READ statement was unsuccessfully executed as a result of an attempt to read a record when no next logical record exists in the file

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 will 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 *CIS COBOL Operating Guide* specific to your operating system 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.

Status Key 1	Status Key 2
	No Further Information (0)
Successful Completion (0)	X
At End (1)	X
Permanent Error (3)	X
Implementor Defined (9)	O/S Error Number

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.

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. (See also Appendix I in this manual).

General Format

```
FILE-CONTROL. file-control-entry ...
```

The FILE CONTROL Entry

Function

The file control entry names a file and may specify other file-related information.

General Format

```
SELECT file-name
ASSIGN TO { external-file-name-literal | file-identifier } [ , external-file-name-literal | file-identifier ]
[ ; ORGANIZATION IS SEQUENTIAL | LINE SEQUENTIAL ]
[; ACCESS MODE IS SEQUENTIAL]
[; FILE STATUS IS data-name-1]
```

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.

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

General Rules

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

2. The ORGANIZATION clause specifies the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed.

3. When LINE SEQUENTIAL ORGANIZATION is specified, the file is treated as consisting of variable length records, each record containing one line of data. The definition of a line of data varies with different operating systems. Some terminate line "records" with the Carriage Return and Line Feed characters, or one of them, and some pad out as fixed length records. CIS COBOL therefore is always compatible with the Editor software in any Operating System in this respect.

4. Records in the file are accessed in the sequence dictated by the file organization. This sequence is specified by predecessor-successor record relationships established by the execution of WRITE statements when the file is created or extended.

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-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 the section called “I-O Status” in this Chapter).

The I-O-CONTROL Paragraph

Function

The I-O CONTROL paragraph specifies the points at which re-run is to be established, the memory area which is to be shared by different files, and the location of files on a multiple file reel.

General Format

```
I-O-CONTROL. [ ; RERUN [ ON { file-name-1 | implementor-name } ] EVERY {{ [END OF] { REEL | UNIT } | integer-1 RECORDS } | OF file-name-2 | integer-2 CLOCK-UNITS | condition-name } ]... [ ; SAME AREA FOR file-name-3 [, file-name-4]... ]... .
```

Syntax Rules

1. The I-O-CONTROL paragraph is optional. The whole clause is for documentation only when present.
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 and is for documentation purposes only.
4. When either the integer-1 RECORDS clause or the integer-2 CLOCK-UNITS clause is specified, implementor-name must be given in the RERUN clause.
5. More than one RERUN clause may be specified for a given file-name-2.
6. The files referenced in the SAME AREA clause 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 is treated as for documentation purposes only.

DATA DIVISION IN THE SEQUENTIAL I-O MODULE

FILE SECTION

In a CIS COBOL program the file description entry (FD) represents the highest level of organisation 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, 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 CONCEPT OF LEVELS in Chapter 2, while the elements allowed in a record description are shown in the the section called “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.

General Format

```
FD file-name [ ; BLOCK CONTAINS integer-2 { RECORDS | CHARACTERS } ]
[ ; RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS ]
[ ; LABEL { RECORD IS | RECORDS ARE } { STANDARD | OMITTED } ]
[ ; VALUE OF data-name-1 IS literal-1 [, data-name-2 IS literal-2]... ] [ ; DATA { RECORD IS |
RECORDS ARE } data-name-3 [, data-name-4] ]
[; CODE-SET IS alphabet-name]
```

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 when the ANSI switch is unset (See Chapter 2).
3. One or more record description entries must follow the file description entry.

THE BLOCK CONTAINS CLAUSE

Function

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

General Format

```
BLOCK CONTAINS integer { 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
```

Syntax Rules

1. When the CODE-SET clause is specified for a file, all data in that file must be described as usage is DISPLAY and any signed numeric data must be described with the SIGN IS SEPARATE clause.
2. The alphabet-name clause referenced by the CODE-SET clause must not specify the literal phrase.
3. The CODE-SET clause may only be specified for non-disk files.

General Rule

The CODE-SET clause is specified for documentation purposes only.

THE DATA RECORDS CLAUSE

Function

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

General Format

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

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 | RECORDS ARE } { STANDARD | OMITTED }

Syntax Rule

This clause is required in every file description entry, when the ANSI switch is set.

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.

THE VALUE OF CLAUSE

Function

The VALUE OF clause specifies the description of an item in the label records associated with a file.

General Format

VALUE OF data-name-1 IS literal-1 [, data-name-2 IS literal-2]...

General Rules

1. This clause is used for documentation purposes only.
2. 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 files.

General Format

CLOSE file-name-1 { REEL | UNIT }

Syntax Rule

The REEL or UNIT phrase must only be used for sequential files, and are for documentation purposes only.

General Rules

1. A CLOSE statement may only be executed for a file in an open mode.
2. The action taken if the 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 leave the file open.
3. 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.
4. 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.

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

```
OPEN { INPUT file-name-1 [, file-name-2]... OUTPUT file-name-3 [, file-name-4]... I-O file-name-5
[, file-name-6]... EXTEND file-name-7 [, file-name-8]... }
```

Syntax Rules

1. The I-O phrase can only be used for disk files, except for files in line sequential organization.
2. The EXTEND phrase can only by used for sequential files and line sequential files.

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 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 5-1, '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 5.1. Permissable Combinations of Statements and OPEN Modes for Sequential I/O.

Statement	Open Mode			
READ	Input	Output	Input-Output	Extend
	X		X	
WRITE		X		X
REWRITE			X	
This OPEN mode is not supported for ORGANIZATION line sequential files.				

5. A file may be opened with the INPUT, OUTPUT, EXTEND and I-O phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement 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.
   b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the ASSIGNed name to be written in accordance with the operating system conventions for opening files for output.
8. The file description entry for file-name-1, file-name-5, must be equivalent to that used when this file was created.
9. If the storage medium for the file permits rewinding, execution of the OPEN statement causes the file to be positioned at its beginning.

10. 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.

11. 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.

12. The I-O phrase permits the opening of a disk for both input and output operation except for file in ORGANIZATION LINE SEQUENTIAL.

13. 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.

THE READ STATEMENT

Function

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

General Format

READ file-name RECORD [INTO identifier] [; 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.

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 the section called “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. 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 READ statement, the position of current record pointer for that file is undefined, the execution of that READ statement is unsuccessful.

9. If the end of a reel or unit is recognized during the execution of a READ statement, an end-of-file status condition exists.

   a. The standard ending reel/unit label procedure.

   b. A reel/unit swap.

   c. The standard beginning reel/unit label procedure.

   d. The first data record of the new reel/unit is made available.

10. 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 the section called “I-O Status”).

11. 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 the section called “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.

12. 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.

13. 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.

4. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area,

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 the section called “I-O Status” in this Chapter).

8. The REWRITE statement cannot be used with line sequential files.

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 | ERROR } PROCEDURE ON { *file-name-1* | INPUT | OUTPUT | I-O | EXTEND }

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 procedure to be used.

2. The USE statement itself is never executed; it merely defines the conditions calling for the execution of the USE procedures.

General Rules

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

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.

THE WRITE STATEMENT

Function

The WRITE statement releases a logical record for an output file. It can also be used for vertical positioning of lines within a logical page.

General Format

`WRITE` record-name [FROM identifier-1] [BEFORE | AFTER ADVANCING integer LINE | LINES | TAB | PAGE]

Syntax Rules

1. Record-name and identifier-1 must not reference the same storage area.

2. When TAB is specified the result is to cause the paper to throw to the standard vertical tabulation position.

3. The record-name is the name of a logical record in the File Section cf the Data Division.

4. Integer may be zero.

General Rules

1. The associated file must be open in the OUTPUT 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 execution of the WRITE statement was unsuccessful due to a boundary violation.

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.

   After execution of the WRITE statement is complete, the information in the area referenced by identifier-1 is available, even though the information in the area referenced by record-name may not be. (See general rule 2.)

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 the section called “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 disk 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.

9. The ADVANCING phrase allows control of the vertical positioning of each line on a representation of a printed page.

   a. With ORGANIZATION SEQUENTIAL if the ADVANCING phrase is not used, automatic advancing is provided when output is directed to a list-device to act as if the user had specified AFTER ADVANCING 1 LINE. If the ADVANCING phrase is used, advancing is provided as follows:

      i. If integer is specified, the representation of the printed page is advanced the number of lines equal to the value of integer.

      ii. If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced.

      iii. If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced.

      iv. 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.

   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(iv) above.

      If the ADVANCING phrase is used or the output is directed to a list device, the resulting file is restricted in its use. general, the file cannot be read to automatically retrieve the logical records written. In particular, if the BEFORE ADVANCING and AFTER ADVANCING clauses are both used (implicitly or explicitly) when writing the file, it may not be opened as an input file with ORGANIZATION LINE SEQUENTIAL.

10. When an attempt is made to write beyond the externally defined boundaries of a sequential file, an exception condition exists and the contents of the record area are unaffected. The following action takes place:

a. The value of the FILE STATUS data item, if any, of the associated file is set to a value indicating a boundary violation. (See the section called “I-O Status” in this Chapter).

b. If a USE AFTER STANDARD EXCEPTION declarative is explicitly or implicitly specified for the file, that declarative procedure will then be executed.

c. If a USE AFTER STANDARD EXCEPTION declarative is not explicitly or implicitly specified for the file, the result is undefined.

Chapter 6. RELATIVE INPUT AND OUTPUT

INTRODUCTION TO THE RELATIVE I-O MODULE

The Relative I-O module provides a capability to access 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. (See the *CIS COBOL Operating Guide* for the maximum number of records in a relative 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. Records are stored and retrieved based on this number. For example, the tenth record is the one addressed by relative record number 10 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. In the dynamic access mode, the programmer may change at will from sequential access to random access using appropriate forms of input-output statements.

Current Record Pointer

The current record printer is a conceptual entity used in this document to facilitate specification of the next record to be accessed within a given file. The concept of the current record pointer has no meaning for a file opened in the output mode. The setting of the current record pointer is affected only by the OPEN, START and READ statements.

I-O Status

If the FILE STATUS clause is specified in a file control entry, a value is placed into the specified two-character data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE, DELETE or START statement and before any applicable USE procedure is executed, to indicate to the COBOL program the status of that input-output operation.

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' - indicates Successful Completion
'1' - indicates At End
'2' - indicates Invalid Key

'3' - indicates Permanent Error
'9' - indicates an 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:
- Duplicate Key
- No Record Found
- Boundary Violation

3 Permanent Error. The input-output statement was unsuccessfully executed 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 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 Operation 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 *CIS 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 values of status key 1 and status key 2 are shown in the table. An 'X' at an intersection indicates a valid permissible combination.

Status Key 1	Status Key 2			
	No Further Information (0)	Duplicate Key (2)	No Record Found (3)	Boundary Violation (4)
Successful Completion (0)	X			
At End (1)	X			

Invalid Key (2)		X	X	X
Permanent Error (3)	X			
Implementor Defined (9)	Operating System Error Message Number			

The INVALID KEY Condition

The INVALID KEY condition can occur as a result of the execution of a START, READ, WRITE, REWRITE or DELETE statement. For details of the causes of the condition, see The START Statement, The READ Statement, The WRITE Statement, The REWRITE Statement, and The DELETE Statement later in this chapter.

When the INVALID KEY condition is recognised, 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 the section called “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 recognised the condition is unsuccessful, and the file is not affected.

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.

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 Appendix F in the *CIS COBOL Operating Guide*).

General Format

```
FILE-CONTROL {file-control-entry}...
```

The File-Control Entry

Function

The file control entry names a file and may specify other file-related information.

General Format

```
SELECT file-name
ASSIGN TO { external-file-name-literal | file-identifier } [ , { external-file-name-literal | file-identifier } ]
; ORGANIZATION IS RELATIVE
[ ; ACCESS MODE IS { SEQUENTIAL ,RELATIVE KEY IS data-name | { RANDOM | DYNAMIC } ,RELATIVE KEY IS data-name } ]
[; FILE STATUS IS data-name-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-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, the Report Section, or the Communication Section.

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

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

General Rules

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

2.

3. When the access mode is sequential , records in the file are accessed in the sequence dictated by the file organization. This sequence is the order of ascending relative record numbers of existing records in the file.

4. When the FILE STATUS clause is specified, a value will be moved by the operating system into the data item specified by data-name-2 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 the section called “I-O Status” in this Chapter).

5. If the access mode is random, the value of the RELATIVE KEY data item indicates the record to be accessed.

6. When the access made is dynamic, records in the file may be assessed sequentially and/or randomly. (See General Rules 3 and 5) .

7. All records stored in a relative file are uniquely identified by relative record numbers. The relative record number of a given record specifies the record's logical ordinal position in the file. The first logical record has a relative record number of 1, and subsequent logical records have relative record numbers of 2, 3, 4, ...

8. The data item specified by data-name-1 is used to communicate a relative record number between the user and the Operating System.

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

```
I-O-CONTROL. [ ; RERUN [ ON { file-name-1 | implementor-name } ] EVERY { integer-1 RECORDS OF file-name-2 | integer-2 CLOCK-UNITS | condition-name } ]... [ ; SAME AREA FOR file-name-3 [, file-name-4]... ]... .
```

Syntax Rules

1. The I-O-CONTROL paragraph is optional. The whole clause is for documentation purposes only.
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, implementor-name must be given in the RERUN clause.
4. More than one RERUN clause may be specified for a given file-name-2, subject to the following restriction:

   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.
6. More than one SAME clause may be included in a program but file-name must not appear in more than one SAME AREA clause.
7. The files referenced in the SAME AREA clause 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 is treated as for documentation purposes only.

DATA DIVISION IN THE RELATIVE I-O MODULE

FILE SECTION

In a CIS COBOL program the file description entry (FD) represents the highest level or 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 section called “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.

General Format

```
FD file-name [ ; BLOCK CONTAINS integer-2 { RECORDS | CHARACTERS } ]
[ ; RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS ]
[ ; LABEL { RECORD IS | RECORDS ARE } { STANDARD | OMITTED } ]
[ ; VALUE OF implementor-name-1 IS literal-1 [, implementor-name-2 IS literal-2]... ] [ ; DATA
{ RECORD IS | RECORDS ARE } data-name-3 [, data-name-4] ]
```

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 cases, and their order of appearance is immaterial.
3. One or more record description entries must follow the file description entry.

THE BLOCK CONTAINS CLAUSE

Function

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

General Format

```
BLOCK CONTAINS integer-2 { RECORDS | CHARACTERS }
```

General Rules

THE DATA RECORDS CLAUSE

Function

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

General Format

```
DATA { RECORD IS | RECORDS ARE } data-name-1 [, data-name-2]
```

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 | RECORDS ARE } { STANDARD | OMITTED }

Syntax Rule

This clause is required in every file description entry, when the ANSI switch is set.

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.

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 VALUE OF CLAUSE

Function

The VALUE of clause specialises the description of an item in the label records associated with a file.

General Format

VALUE OF data-name-1 IS literal-1 [, data-name-2 IS literal-2]...

Syntax Rules

1. Data-name-1, data-name-2, etc, 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-1, data-name-2 etc, must be in the Working-Storage Section

General Rules

1. This clause is used for documentation purposes only.

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

PROCEDURE DIVISION IN THE RELATIVE I-O MODULE

THE CLOSE STATEMENT

Function

The CLOSE statement terminates the procession of files. The LOCK is for documentation purposes only.

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. 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.

3. 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.

4. 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.

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 the 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.

3. For a file in random or dynamic access mode, the Operating System logically removes from the file that record identified by the contents of the RELATIVE KEY data item associated with file-name. If the file does not contain the record specified by the key, an INVALID key condition exists. (See the section called “The INVALID KEY Condition” in this Chapter).

4. After the successful 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 the section called “I-O Status” in this chapter.

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

OPEN { INPUT *file-name-1* [, *file-name-2*]... | 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 an 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 6-1, 'X' at an intersection indicates that the specified statement, used in the access mode given for that row, may be used with the relative file organization and the open mode given at the top of the column.

Table 6.1. 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 sane 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 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.

8. 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 Format 1 READ statement for the file will result in an AT END condition. If the file does not exist, INPUT will cause an error status.

9. 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. In sequential access mode it will then be used for input; any attempt to WRITE to it will cause an error.

10. Upon successful execution of an OPEN statement with the OUTPUT phrase specified, a file is created. At the time the associated file contains no data. If a file of the same number exists it will be deleted. If write protected, an error status occurs.

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 disk file.

General Format

Format 1

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

Format 2

READ file-name RECORD [INTO identifier] [;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 record area associated with file-name must not be the same storage area.

2. Format must be used (without the NEXT phrase) for all files in sequential access mode.

3. The NEXT phrase must be specified for files in dynamic access mode, when records are to be retrieved sequentially.

4. Format 2 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.

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 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.

   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.

3. The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with filename to be updated. (See the section called “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 I-O 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 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 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 the section called “I-O Status” in this Chapter).

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 the section called “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 recognised, a Format 1 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 READ statement for that file.

13. For a file for which dynamic access mode is specified, a Format 1 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 READ statement updates the contents of the RELATIVE KEY data item such that it contains the relative record number of the record made available.

15. The execution of a Format 2 READ statement sets the current record pointer to, and makes available, the record whose relative record number is contained in the data item named in the RELATIVE KEY phrase for the file. If the file does not contain such a record, the INVALID KEY condition exists and execution of the READ statement is unsuccessful. (See the section called “The INVALID KEY Condition” in this Chapter).

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.
3.

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.
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.
4. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area.
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 the section called “I-O Status” in this Chapter).
8. For a file accessed in either random or dynamic access mode, the Operating System logically replaces the record specified by the contents of the RELATIVE KEY data item associated with the file. If the file does not contain the record specified by the key, the INVALID KEY condition exists. (See the section called “The INVALID KEY Condition” in this Chapter). The updating operation does not take place and the data in the record area is unaffected.

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 | IS = | IS GREATER THAN | IS > | IS NOT LESS THAN | IS NOT < data-name

[;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).
2. If the KEY phrase is not specified the relational operator 'IS EQUAL TO' is implied.
3. The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file-name and a data item as specified in general Rule 5.
   a. The current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.
   b. If the comparison is not sattified by any record in the file, an INVALID KEY condition exists, the execution of the START statement is unsuccessful, and the position of the current record pointer is undefined. (See the section called “The INVALID KEY Condition” in this Chapter).
4. The execution of the START statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. (See the section called “I-O Status” in this Chapter).
5. The comparison described in general rule 3 uses the data item referenced by the RELATIVE KEY clause associated with file-name.

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 | ERROR } PROCEDURE ON { file-name-1 | 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.

General Rules

1. If the INVALID KEY or AT END phrases have not been specified in the input-output statement, 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.

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.

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.

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 Chapter).

2. The logical record released by the execution of the WRITE statement is no longer available in the record area unless the execution of the WRITE statement is unsuccessful due to an INVALID KEY condition.

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 the section called “I-O Status” in this Chapter).

6. The maximum record size for a file ts 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.

9. When a file is opened in the output mode, records may be placed into the file by one of the following:

   a. If the access mode is sequential, the WRITE statement will cause a record to be released to the Operating System. The first record will have a relative record number of one and subsequent records released will have relative record numbers of 2, 3, 4. If the RELATIVE KEY data item has been specified in the file control entry for the associated file, the relative record number of the record just released will be placed into the RELATIVE KEY data item by the Operating System during execution of the WRITE statement.

   b. If the access mode is random or dynamic, prior to the execution of the WRITE statement the value of the RELATIVE KEY data item must be initialised in the program with the relative record number of be associated with the record in the record area. That record is then released to the Operating System by execution of the WRITE statement.

10. When a file is opened in the I-O mode and the access mode is random or dynamic, records are to be inserted in the associated file. The value of the RELATIVE KEY data item must be initialised by the program with the relative record number to be associated with the record in the record area. Execution of a WRITE statement then causes the contents of the record area to be released to the Operating System.

11. The INVALID KEY condition exists under the following circumstances:

    a. When the access mode is random or dynamic, and the RELATIVE KEY data item specifies a record which already exists in the file, or

    b. When an attempt is made to write beyond the externally defined boundaries of the file.

12. When the INVALID KEY condition is recognised, the execution of the WRITE statement is unsuccessful, the contents of the record area are unaffected, and the FILE STATUS data item, if any, 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 in the section called “The INVALID KEY Condition” in this Chapter (see also the section called “I-O Status” in this Chapter).

Chapter 7. INDEXED INPUT AND OUTPUT

INTRODUCTION TO THE INDEXED I-O MODULE

The Indexed I-O module provides a capability to access 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 key 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 includes a key data item, which is associated with an index. The 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.

The data item named in the RECORD KEY clause of the file control entry for a file is the 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 record key. This value must, therefore, be unique and must not be changed when updating the record. The key length must not exceed 32 bytes. See the *CIS COBOL Operating Guide* for the maximum records in a file.

Access Modes

In the sequential access mode, the sequence in which records are accessed is the ascending order of the record key values.

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

In the dynamic access mode, the programmer may change at will from sequential access to random access using appropriate forms of input-output statements.

Current Record Pointer

The current record pointer is a conceptual entity used in this document to facilitate specification of the next record to be accessed within a given file. The concept of the current record pointer has no meaning for a file opened only in the output mode. The setting of the current record pointer is affected only by the OPEN, START and READ statements.

I-O Status

If the FILE STATUS clause is specified in a file control entry, a value is placed into the specified two-character data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE, DELETE or START statement and before any applicable USE procedure is executed, to indicate to the COBOL program the status or that input-output operation.

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 unsuccessfully executed 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 will 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 '2' indicating an INVALID KEY condition, status key 2 contains values to designate the cause of that condition as follows:

1 - Indicates a sequence error for a sequentially accessed indexed file. The ascending sequence requirements of successive record key values have been violated (see The WRITE Statement later in this Chapter), or the record key value has been changed by the COBOL program between the successful execution of a READ statement and the execution of the next REWRITE statement for that file.
2 - Indicates a duplicate key value. An attempt has been made to write a record that would create a duplicate key in an indexed 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 indexed file. This is normally treated as a fatal error by Operating Systems.

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 *CIS 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				

The INVALID KEY Condition

The INVALID KEY condition can occur as a result of the execution of a START, READ, WRITE, REWRITE or DELETE statement. For details of the causes of the condition see THE START STATEMENT, THE READ STATEMENT, THE WRITE STATEMENT, and THE DELETE STATEMENT later in this Chapter.

When the INVALID KEY condition is recognised, 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 the section called “I-O Status”).

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

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.

ENVIRONMENT DIVISION IN THE INDEXED 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 appendix F in the *CIS COBOL Operating Guide*).

General Format

```
FILE-CONTROL {file-control-entry}...
```

The File Control Entry

Function

The file control entry names a file and may specify other file-related information.

General Format

```
SELECT file-name
ASSIGN TO { external-file-name-literal | file-identifier } [ , { external-file-name-literal | file-identifier } ]
; ORGANIZATION IS INDEXED
[ ; ACCESS MODE IS { SEQUENTIAL | RANDOM | DYNAMIC } ]
; RECORD KEY IS data-name-1
[; FILE STATUS IS data-name-3]
```

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 1:1 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-3 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.

5. The data items referenced by data-name-1 must each be defined as a data item of the category alphanumeric within a record description entry associated with that file-name.

General Rules

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

2.

3. 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.

4. When the FILE STATUS clause is specified, a value will be moved by the operating system into the data item specified by data-name-3 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 the section called “I-O Status” in this Chapter).

5. If the access mode is random, the value of the record key data item indicates the record to be accessed.

6. When the access mode is dynamic, records in the file may be accessed sequentially and/or randomly. (See general rules 4 and 6).

7.

8. The data description of data-name-1 as well as relative locations within a record must be the same as that used when the file was created.

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

```
I-O-CONTROL. [ ; RERUN [ ON { file-name-1 | implementor-name } ] EVERY { integer-1
RECORDS OF file-name-2 | integer-2 CLOCK-UNITS | condition-name } ]...
[ ; SAME AREA FOR file-name-3 [, file-name-4]... ]... .
```

Syntax Rules

1. The I-O-CONTROL paragraph is optional. The whole clause is for documentation purposes only when present.

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, implementor-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.

6. More than one SAME clause (SAME AREA) may be included in a program but a file-name must not appear in more than one SAME AREA clause.

7. The files referenced in the SAME AREA clause 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 is treated as for documentation purposes only.

DATA DIVISION IN THE INDEXED I-O MODULE

FILE SECTION

In a COBOL program the file description entry (FD) represents the highest level of organisation 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 section called “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.

General Format

FD file-name [; BLOCK CONTAINS integer-2 { RECORDS | CHARACTERS }]
[; RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS]
[; LABEL { RECORD IS | RECORDS ARE } { STANDARD | OMITTED }]
; VALUE OF data-name-1 IS literal-1 [, data-name-2 IS literal-2]... [; DATA { RECORD IS | RECORDS ARE } data-name-3 [, data-name-4]... .]

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 when the ANSI switch is unset.
3. One or more record description entries must follow the file description entry.

THE BLOCK CONTAINS CLAUSE

Function

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

General Format

BLOCK CONTAINS integer-2 { RECORDS | CHARACTERS }

General Rule

The clause is required for documentation purposes only.

THE DATA RECORDS CLAUSE

Function

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

General Format

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

Syntax Rules

Data-name-1 and data-name-2 are the names of data records and must 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 | RECORDS ARE } { STANDARD | OMITTED }

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.

THE VALUE OF CLAUSE

Function

The VALUE OF clause specialises the description of an item in the label records associated with a file.

General Format

VALUE OF data-name-1 IS literal-1 [, data-name-2 IS literal-2]...

General Rules

1. This clause is used for documentation purposes only.

2. 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. The LOCK phrase is for documentation purposes only.

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. 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.

3. 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.

4. 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.

THE DELETE STATEMENT

Function

The DELETE statement logically removes a record from a file.

General Format

DELETE file-name RECORD [;INVALID KEY imperative-statement]

Syntax Rules

1. The INVALID KEY phase 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 record that was accessed by that READ statement is logically removed from the file.

3. For a file in random or dynamic access mode, the record identified by the contents of the record key data item associated with file-name is logically removed from the file. If the file does not contain the record specified by the key, an INVALID KEY condition exists. (See the section called “The INVALID KEY Condition” in this Chapter).

4. After the successful 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 file-name to be updated. (See the section called “I-O Status” in this Chapter).

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

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

Syntax Rules

1. 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 7-1, 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 organisation and the open mode given at the top of the column.

Table 7.1. Permissible Combinations of Statements and Open Modes for Indexed 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 OPEN statement 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.

   b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the assigned name to be written in accordance with the opera ting system conventions for opening files for output.

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. If no records exist in the file, the current record pointer is set such that the next executed Format 1 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 out put operations. If the file does not exist, it will be created.

11. 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.

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] [; AT END imperative-statement]

Format 2

READ file-name RECORD [INTO identifier] [;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.
2. Format 1 must be used (without the NEXT phrase) for all files in sequential access mode.
3. Format 2 is used for files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.
4. The NEXT phrase must be specified for files in dynamic access mode, when records are to be retrieved sequentially.
5.

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 a Format 1 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 key sequence and that record is then 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 in key sequence 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 the section called “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. 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 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 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 the section called “I-O Status” in this Chapter). -

10. When the AT END condition is recognised 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 the section called “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 recognised, a Format 1 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 READ statement for that file.

13. For a file which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from that file as described in general rule 2 above.

14. Execution of a Format 2 READ statement causes the value of the key that to be compared with the value contained in the corresponding data item of the stored records in the file, until the first record has 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 section called “The INVALID KEY Condition” in this Chapter).

THE REWRITE STATEMENT

Function

The REWRITE statement logically replaces a record existing in a mass storage 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.
3. The INVALID KEY phrase must be specified in the REWRITE statement for files 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.
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.
4. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area.
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 the section called “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 record key. When the REWRITE statement is executed the value contained in the record key data item of the record to be replaced must be equal to the value of the 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 record key data item.
10. The INVALID KEY condition exists when:
    a. The access mode is sequential and the value contained in the record key data item of the record to be replaced is not equal to the value of the record key of the last record read from this file or,
    b. The value contained in the record key data item does not equal that of any record stored in the file, or

c. The updating operation does not take place and the data in the record area is unaffected. (See the section called “The INVALID KEY Condition” in this Chapter).

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 | IS = | IS GREATER THAN | IS > | IS NOT LESS THAN | IS NOT < data-name
[;INVALID KEY imperative-statement]]

NOTE: The required relational characters '>', '<' 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. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.

4. If file-name is the name of an indexed file, and if a KEY phrase is specified, data-name may reference a data item specified as the record key associated with file-name, or it may reference any data item of category alphanumeric subordinate to the data-name of a data item specified as the record key associated with file-name whose leftmost character position corresponds to the leftmost character position of that record key data item.

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).

2. If the KEY phrase is not specified, 'IS EQUAL TO' is implied.

3. The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file-name and a data item as specified in general rule 5. If file-name references an indexed file and the operands are of unequal size, comparison proceeds as though the longer one were truncated on the right such that its length is equal to that of the shorter. All other nonnumeric comparison rules apply except that the presence of the PROGRAM COLLATING SEQUENCE clause will have no effect on the comparison. (See Comparison of Nonnumeric Operands).

   a. The current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

   b. If the comparison is not satisfied by any record in the f11.e, an INVALID KEY condition exists, the execution of the START statement is unsuccessful, and the position of the current record pointer is undefined. (See the section called “The INVALID KEY Condition” in this Chapter)

4. The execution of the START statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. (See the section called “I-O Status”).

5. If the KEY phrase is specified, the comparison described in general rule 3 uses the data item referenced by data-name.

6. If the KEY phrase is not specified, the comparison described in general rule 3 uses ti1e data item referenced in the RECORD KEY clause associated with file-name.

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 | ERROR } PROCEDURE ON { file-name-1 | 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; i t merely defines the conditions calling for the execution of the USE procedures.

General Rules

1. If the INVALID KEY phrase on the AT END phrase have not been specified in the input-output statements the designated procedures are executed by the input-output system after completing the standard input-output routine upon recognition of the INVALID KEY or AT END condition.

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 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.

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.

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 execution of the WRITE statement is unsuccessful due to an INVALID KEY condition.

3. The results of the execution of the WRITE statement with the FROM phrase is equivalent tn 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 the section called “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.

9. Execution of the WRITE statement causes the contents of the record area to be released. The Operating System utilizes the content of the record key in such a way that subsequent access of the record may be made based upon the specified record key.

10. The value of the record key must be unique within the records in the file.

11. The data item specified as the 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 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. 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 record key is not greater than the value of the 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 record key is equal to the value of a record key of a record already existing in the file, or

    c. When an attempt is made to write beyond the externally defined boundaries of the file.

15. When the INVALID KEY condition is recognised 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 section called “The INVALID KEY Condition” (See also the section called “I-O Status” in this Chapter).

Chapter 8. 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 sections with the same segment-number must be contiguous in the source program. All segments specified as permanent segments must be continuous in the source program.

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.

ORGANIZATION

Program Segments

Although it is not mandatory, the Procedure Division for a source program is usually written as a consecutive group of sections, each of which is composed of a series of closely related operations that are designed to collectively perform a particular function. However, when segmentation is used, the entire Procedure Division must be in sections. In addition, each section must be classified as belonging either to the fixed portion or to one of the independent segments of the object program.

Fixed Portion

The fixed portion is defined as that part of the object program which is logically treated as if it were always in memory. This portion of the program is composed of fixed permanent segments.

A fixed permanent segment is a segment in the fixed portion which cannot be overlaid by any other part of the program.

Independent Segments

An independent segment is defined as part of the object program which can overlay, and can be overlaid by 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 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 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.

2. Frequency of Use - Generally, the more frequently a section is referred to, the lower its segment-number, the less frequently it is referred to, the higher its segment-number,

3. Relationship to Other Sections - Sections which frequently communicate with one another should be given the same segment-numbers

SEGMENTATION CONTROL

The logical sequence of the program is the same as the physical sequence except for specific transfers of control. Control may be transferred within a source program to any paragraph in a section; that is, it is not mandatory to transfer control to the beginning of a section.

STRUCTURE OF PROGRAM SEGMENTS

SEGMENT-NUMBERS

Section classification is accomplished by means of a system of segment-numbers. The segment-number is included in the section header.

GENERAL FORMAT

section-name SECTION [segment-number]

SYNTAX RULES

1. The segment-number must be an integer ranging in value from 0 through 99.

2. If the segment-number is omitted from the section header, the segment-number is assumed to be 0.

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 must be together in the source program.

2. Segments with segment-number 0 through 49 belong to the fixed portion of the object program. All sections with segment-number 0 through 49 must be together in the source program.

3. Segments with segment-number 50 through 99 are independent segments.

RESTRICTIONS ON PROGRAM FLOW

When segmentation is used, the following restrictions are placed on the ALTER and PERFORM statement.

THE ALTER STATEMENT

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.

THE PERFORM STATEMENT

A PERFORM statement that appears in a section that is not 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:

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

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.

EXTRA INTERMEDIATE CODE FILES

When segmentation is used, extra intermediate code files are generated by the compiler as follows:

filename.Inn - Intermediate code files one for each independent segment

filename.ISR - Inter-Segment Reference table one per segmented program

filename.Dnn - Dictionary files one for each independent segment except the last

where:

filename is the name without the extension of the principal intermediate code file

nn is a segment number that identifies the particular segment

Note

The filename.Dnn files are written and used solely by the compiler, and need not be retained after compilation. The filename.Inn files and the filename.ISR file must be retained as part of the object program and must also be copied when the program is copied.

Chapter 9. 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.

CIS 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.

THE COPY STATEMENT

FUNCTION

The COPY statement incorporates text into a CIS COBOL source program.

GENERAL FORMAT

COPY text-name | external-file-name-literal .

SYNTAX RULES

1. Text-name defines a unique external file name which conforms to the rules for COBOL user-defined words. In a text-name lower case is translated into upper case. External-file-name-literal is an alphanumeric literal enclosed in quotes that conforms to the operating system rules for filenames.

2. The COPY statement must be preceded by a space and terminated by the separator period.

3. 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.

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. The library text is copied unchanged.

4. If the unit identifier is not explicitly specified, default is to the drive from which the compiler is loaded.

5. The text produced as a result of the complete processing of a COPY statement must not contain a COPY statement.

Chapter 10. 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 CIS COBOL Run-Time Debug Package is an extension to ANSI COBOL that provides break-point facilities in the user's program. Programs may be run from the start until a specified break-point is reached, when control is passed back to the user. At this point, data areas may be inspected or changed.

CIS COBOL RUN-TIME DEBUG EXTENSION

The Run-Time debug is entered as an option by the user and the user program is then tested line by line, paragraph by paragraph and so on as required. The commands to the package can reference procedure statements and data areas by means of a 4-digit hexadecimal code output by the compiler against each line of the compilation listing. Powerful macros of commands can be used to give very sophisticated debugging facilities. The precise details for using the package vary according to the host operating system, and are therefore contained in the *CIS COBOL Operating Guide* for your Operating System.

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.

The reserved word DEBUG-ITEM is the name for a special register generated automatically by the compiler that supports the debugging facility. Only one DEBUG-ITEM is allocated per program. The names of the subordinate data items in DEBUG-ITEM are also reserved words.

COMPILE TIME SWITCH

The DEBUGGING MODE clause is written as part of the SOURCE-COMPUTER paragraph in the Environment Division. It serves as a compile-time switch over debugging statements written in the program.

When DEBUGGING MODE is not specified in a program, all the debugging lines are compiled as if they were comment lines and their syntax is not checked.

COBOL DEBUG OBJECT TIME SWITCH

An object time switch dynamically activates the debugging code inserted by the compiler. This switch cannot be addressed in the program; it is controlled outside the COBOL environment. If the switch

is 'on', the effects of any USE FOR DEBUGGING statements written in the source program are permitted. If the switch is 'off', all the effects described in the USE FOR DEBUGGING Statement, are inhibited. Recompilation of the source program is not required to provide or take away this facility.

The object time switch has no effect on the execution of the object program if the WITH DEBUGGING MODE clause was not specified in the source program at compile time. The switch is described in the *CIS COBOL Operating Guide*.

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 lines.

PROCEDURE DIVISION IN COBOL DEBUG

The USE FOR DEBUGGING Statement

Function

The USE FOR DEBUGGING statement identifies the user items that are to be monitored by the associated debugging section.

General Format

```
section-name SECTION [segment number] . USE FOR DEBUGGING ON { procedure-name-1 | ALL
PROCEDURES } [ , { procedure-name-2 | ALL PROCEDURES }... ]
```

Syntax Rules

1. Debugging section(s), if specified, must appear together immediately after the DECLARATIVES header.

2. Except in the USE FOR DEBUGGING statement itself, there must be no reference to any non-declarative procedure within the debugging section.

3. Statements appearing outside of the set of debugging sections must not reference procedure-names defined within the set of debugging sections.

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 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. 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 procedure-name-1, apply equally to procedure-name-2.

2. Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.

3. When procedure-name-1 is specified in a USE FOR DEBUGGING statement that debugging section is executed:
   a. Immediately before each execution of the named procedure;
   b. Immediately after the execution of an ALTER statement which references procedure-name-1.

4. The ALL PROCEDURES phrase causes the effects described in general rule 3 to occur for every procedure-name in the program, except those appearing within a debugging section.

5. 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.

6. A reference to procedure-name-1 as a qualifier does not constitute reference to that item for the debugging described in the general rules above.

7. Associated with each execution of a debugging section is the special register DEBUG-ITEM, which provides information about the conditions that caused the execution of a debugging section. DEBUG-ITEM has the following implicit description:

```
01 DEBUG-ITEM.
   02   DEBUG-LINE      PICTURE IS X(6).
   02   FILLER          PICTURE IS X VALUE SPACE.
   02   DEBUG-NAME      PICTURE IS X(30).
   02   FILLER          PICTURE IS X(19) VALUE SPACE.
   02   DEBUG-CONTENTS  PICTURE IS X(n).
```

8. Prior to each execution of a debugging section, the contents of the data item referenced by DEBUG-ITEM are space-filled. The contents of data items subordinate to DEBUG-ITEM are then updated, according to the following general rules, immediately before control is passed to that debugging section. The contents of any data item not specified in the following general rules remains spaces.

Updating is accomplished in accordance with the rules for the MOVE statement, the sole exception being the move to DEBUG-CONTENTS when the move is treated exactly as if it was an alphanumeric to alphanumeric elementary move with no conversion of data from one form of internal representation to another.

9. The contents of DEBUG-LINE is the relevant COBOL source line number. This provides the means of identifying a particular source statement.

10. DEBUG-NAME contains the first 30 characters of the name that caused the debugging section to be executed.

    Subscripts/indices, if any, are not entered into DEBUG-NAME.

11. DEBUG-CONTENTS is a data item that is large enough to contain the data required by the following general rules.

12. f 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'.

13. 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.

14. 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.

15. If the transfer to control from the control mechanism associated with a PERFORM statement causes the debugging section associated with procedure-name-1 to be executed, the following conditions exist:
    a. DEBUG-LINE identifies the PERFORM statement that references procedure-name-1.
    b. DEBUG-NAME contains procedure-name-1.
    c. DEBUG-CONTENTS contains 'PERFORM LOOP'.

16. If procedure-name-1 is a USE procedure that is to be executed, the following conditions exist:
    a. DEBUG-LINE identifies the statement that causes execution of the USE procedure.
    b. DEBUG-NAME contains procedure-name-1.
    c. DEBUG-CONTENTS contains 'USE PROCEDURE'.

17. 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'.

DEBUGGING LINES

A debugging line is any line with a 'D' in the indicator area of the line. Any debugging line that consists solely of spaces from margin A to margin R is considered the same as a blank line.

The contents of a debugging line must be such that a syntactically correct program is formed with or without the debugging lines being considered as comment lines.

A debugging line will be considered to have all the characteristics of a comment line, if the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph.

Successive debugging lines are allowed. Continuation of debugging lines is permitted, except that each continuation line must contain a 'D' in the indicator area, and character-strings may not be broken across two lines.

A debugging line is only permitted in the program after the OBJECT-COMPUTER paragraph.

Chapter 11. 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.

Data items defined in the Linkage Section of the called program may be referenced within the Procedure Division of the called program only if they are specified as operands of the USING phrase of the Procedure Division header or are subordinate to such operands, and the object program is under the control of a CALL statement that specifies a USING phrase.

The structure of the Linkage Section is the same as that previously described for the Working-Storage Section, beginning with a section header, followed by data description entries for noncontiguous data items and/or record description entries.

Each Linkage Section record-name and noncontiguous item name must be unique within the called program since it cannot be qualified, Data items defined in the Linkage Section of the called program must not be associated with data items defined in the Report Section of the calling program.

Of those items defined in the Linkage Section only data-name-1, data-name-2, ... in the USING phrase of the Procedure Division header, data items subordinate to these data-names, and condition-names and/or index-names associated with such data-names and/or subordinate data items, may be referenced in the Procedure Division.

Noncontiguous Linkage Storage

Items in the Linkage Section that bear no hierarchic relationship to one another need not be grouped into records and are classified and defined as noncontiguous elementary items. Each of these data items is defined in a separate data description entry which begins with the special level-number 77.

The following data clauses are required in each data description entry:

- Level-number 77
- Data-name
- The PICTURE clause or the USAGE IS INDEX clause

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

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.

When the USING phrase is present, the object program operates as if data-name-1 of the Procedure Division header in the called program and data-name-1 in the USING phrase of the CALL statement in the calling program 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. In like manner, there is an equivalent relationship between data-name-2, ..., in the USING phrase of the called program and data-name-2, ..., in the USING phrase of the CALL statement in the calling program. A data-name must not appear more than once in the USING phrase in the Procedure Division header of the called program; however, a given data-name may appear more than once in the same USING phrase of a CALL statement.

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 | literal-1 } [USING data-name-1 [, data-name-2]...] [ON OVERFLOW imperative-statement]

Format 2

```
CALL { literal-2 | identifier-2 } [ USING data-name-3 [, data-name-4]... ]
```

Syntax Rules

1. Literal-1 must be a nonnumeric literal.
2. Identifier-1 must be defined as an alphanumeric data item usage 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.
4. Each of the operands in the USING phrase must have been defined as a data item in the File Section, Working-Storage Section, or Linkage Section, and must have a level-number of 01 or 77.
5. Literal-2 must be a nonnumeric literal.
6. Identifier-2 must must be defined as an alphanumeric data item with a numeric value, e.g. CALL "3" or CALL D-NAM where D-NAM is defined as class alphanumeric, and usage display, containing a numeric value.

General Rules

1. The program whose name is specified by the value of literal-1 or identifier-1 is a called intermediate code module, literal-2 is a called run time subroutine; the program in which the CALL statement appears is the calling program.
2. The execution of a CALL statement causes control to pass to the called program.
3. In format 1, a called intermediate code module is loaded from disk the first time it is called within a run-unit and the first time it is called after a CANCEL to the called program.

   On all other entries into the called program, the state of the program remains unchanged from its state when last executed. This includes all data fields, the status and positioning of all files, and all alterable switch settings.
4. In format 2, a called run time subroutine is always in the state in which it last exited.
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.

THE CANCEL STATEMENT

Function

The CANCEL statement releases the memory areas occupied by the referred to program.

General Format

```
CANCEL { identifier-1 | literal-1 } [ { identifier-2 | literal-2 } ]...
```

Syntax Rules

1. Literal-1, literal-2, ... , must each be a nonnumeric literal.
2. Identifier-1, identifier-2, must each be defined as an alphanumeric data item such that its value can be a program name.

General Rules

1. After the execution of a CANCEL statement, the program referred to ceases to have any logical relationship to the run unit in which the CANCEL statement appears. A subsequently executed CALL statement naming the same program will result in that program being initiated in its initial state. The memory areas associated with the named programs are released so as to be made available for disposition by the operating system.
2. A program named in the CANCEL statement must not refer to any program that has been called and has not yet executed an EXIT PROGRAM statement.
3. A logical relationship to a cancelled subprogram is established only by execution of a subsequent call statement.
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.

THE EXIT PROGRAM STATEMENT

Function

The EXIT PROGRAM statement marks the logical end of a called program.

General Format

```
EXIT PROGRAM
```

Syntax Rules

1. The EXIT PROGRAM statement must appear in a sentence by itself.
2. The EXIT PROGRAM sentence must be the only sentence in the paragraph.

General Rule

An execution of an EXIT PROGRAM statement in a called program causes control to be passed to the calling program. Execution of an EXIT PROGRAM statement in a program which is not called behaves as if the statement were an EXIT statement. (See THE EXIT STATEMENT in Chapter 3) .

Chapter 12. PROGRAMMING TECHNIQUES, USEFUL HINTS AND PROGRAM SIZING

PROGRAMMING TECHNIQUES

Although COBOL is written in an essentially free form, the user will nevertheless obtain many advantages from a few self-imposed disciplines. 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/accepted. Therefore redefine a group of fields as a single field for DISPLAY whenever possible and avoid unnecessary numbers of small fields in 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.

7. CIS COBOL provides for values greater than decimal 99 to be stored in a nonnumeric field of one character, e.g, `PIC X "7F"`

   This is an extension to the ANSI COBOL standard X3.23 (1974). (See under Nonnumeric Literals in Chapter 2).

   Note, however, that the rules for moving such a field comply with the ANSI standard in that the contents will be truncated if over decimal 99.

   If your operating system returns an error number greater than 99 in the error Status Key 2 byte (see I-O Status in chapters 5, 6 and 7) careful redefinition of data-items is required if you wish to display this status with its correct decimal value. See the appendix that describes disk files in your operating system specific *CIS COBOL Operating Guide* for a sample program.

USEFUL HINTS

When writing interactive programs the following facilities of CIS COBOL should be remembered:

1. By use of the CURSOR IS facility and the ACCEPT statement it is easy to program conditionally depending on the cursor position after a menu type of prompt. The operator need then only move the cursor to the option required to reply to the prompt, or just press RETURN in the default case.

2. By use of the ACCEPT FROM CONSOLE facility it is easy to pass parameters to your program via the Run command line. See THE ACCEPT STATEMENT in Chapter 3.

3. Remember always to end your CIS COBOL program with a period. Invalid intermediate code can result if this final period is missing.

4. Note that the data part of an indexed sequential file may be accessed relatively. However, the first record (relatively) is inaccessible. since relative file access begins at record number 1, as specified in the ANSI COBOL standard X3.23(1974).

5. Never define a Linkage Section in the main program, only in sub-programs. The CIS COBOL Compiler will not treat such a Linkage Section as an error but it can result in memory content corruption at run time.

6. Be careful to specify literal filenames in Select statements in quotation marks ("...."). This is the only indication to the compiler that a literal filename is desired.

   (Filename identifiers are not declared in the Working Storage Section or elsewhere explicitly). The omission of quotation marks where required will result in an undefined file being accessed at run time.

Table 12.1. Data Dictionary Entry Sizing

User-defined name	**Number of Bytes[1]**
File-name	18 + n
Record-name	8 + n
Key-name	8 + n
Status-name	8 + n
Paragraph-name	6 + n
Alphanumeric < 32 characters	$8 + n^2$
Alphanumeric $\geq$ 32 characters	$7 + n^2$
Numeric integer	$8 + n^2$
Numeric non integer	$7 + n^2$
Numeric edited	$8 + n^2$
	7 + n + x

1. n = number of characters in user-defined name.

   For a FILLER, n = 1.

   x number of characters in PICture, after coalescing repetitions.

   e.g. 9 9 9 9 . 9 = 3 bytes
   9 (4) . 9 = 3 bytes
   Z (2) 9 (4) . 9 (3)= 4 bytes

2. Subtract 1 byte if item is in the first 256 bytes of Working-Storage.

   Add 4 bytes if item has an OCCURS clause associated with it.

   Add 2 bytes if item is subordinate to an item described with OCCURS.

Appendix A. RESERVED WORD LIST

This appendix contains a full list of COBOL and CIS COBOL reserved words. A shaded reserved word is a CIS COBOL extension to ANSI COBOL.

The / symbol denotes that the text up to that point is a reserved word, as is the whole word.

e.g., In INDEX/ED, INDEX and INDEXED are reserved words IN SPACE/S, SPACE and SPACES are reserved words.

ACCEPT	DYNAMIC	NEGATIVE	SORT
ACCESS	ELSE	NEXT	SORT-MERGE
ADD	END	NOT	SOURCE-COMPUTER
ADVANCING	ENTER	NUMERIC	SPACE/S
AFTER	ENVIRONMENT	OBJECT-COMPUTER	SPECIAL-NAMES
ALL	EQUAL	OCCURS	STANDARD
ALPHABETIC	ERROR	OF	STANDARD-1
ALTER	EVERY	OFF	START
AND	EXCEPTION	OMITTED	STATUS
ARE	EXCESS-3	ON	STOP
AREA	EXCLUSIVE	OPEN	SUBTRACT
ASCENDING	EXIT	OR	SWITCH
ASSIGN	EXTEND	ORGANIZATION	SYNC/HRONIZED
AT	FD	OUTPUT	SYSIN
AUTHOR	FILE	OVERFLOW	SYSOUT
AUTOMATIC	FILE-CONTROL	PAGE	TAB
BEFORE	FILLER	PERFORM	TABLE
BLANK	FIRST	PIC/TURE	TALLYING
BLOCK	FOR	POSITIVE	THAN
BY	FROM	PROCEDURE/S	THEN
CALL	GIVING	PROCEED	THROUGH
CANCEL	GO	PROGRAM	THRU
CHARACTER/S	GREATER	PROGRAM-ID	TIMES
CLOCK-UNITS	HIGH-VALUE/S	QUOTE/S	TO
CLOSE	I-O/-CONTROL	RANDOM	TRAILING
COBOL	IDENTIFICATION	RD	TYPE
CODE-SET	IF	READ	UNIT
COLLATING	INDEX/ED	RECORD/S	UNTIL
COMMA	INITIAL	REDEFINES	UP
COMMIT	INPUT/-OUTPUT	REEL	UPON
COMP-M	INSPECT	RELATIVE	USAGE
COMP-N	INSTALLATION	RELEASE	USE
COMP-3	INTO	REMAINDER	USING
COMP/UTATIONAL/-3	INVALID	REPLACING	VALUE/S
CONFIGURATION	IS	RERUN	VARYING
CONSOLE	JUST/IFIED	RETURN	WHEN
CONTAINS	KEPT	REWRITE	WITH
COPY	KEY	RIGHT	WORDS
CRT	LABEL	ROLLBACK	WORKING-STORAGE
CRT-UNDER	LEADING	ROUNDED	WRITE
CURRENCY	LEFT	RUN	ZERO/ES or S
CURSOR	LESS	SAME	. (period)
DATA	LIMIT/S	SD	(
DATE-COMPILED	LINE/S	SECTION	-
DATE-WRITTEN	LINKAGE	SECURITY	)
DEBUGGING	LOCK	SEGMENT	;
DECIMAL-POINT	LOW-VALUE/S	SEGMENT-LIMIT	+

DECLARATIVES	MANUAL	SELECT	,
DELETE	MEMORY	SENTENCE	<
DEPENDING	MERGE	SEPARATE	=
DESCENDING	MODE	SEQUENCE	>
DISPLAY	MODULES	SEQUENTIAL	/
DIVIDE	MOVE	SET	*
DIVISION	MULTIPLY	SIGN	
DOWN	NATIVE	SIZE	

Note that the Level II COBOL product contains the following additional reserved words. If you wish to ensure that your CIS COBOL programs are upward compatible with Level II COBOL do not use these words as user-names.

ALSO	DELIMITED	LINAGE/-COUNTER	SEARCH
ALTERNATE	DELIMITER	MESSAGE	SEND
BOTTOM	DESTINATION	MULTIPLE	STRING
COMMUNICATION	DISABLE	NO	SUB-QUEUE-1
COMPUTE	DUPLICATES	OPTIONAL	SUB-QUEUE-2
CORR/ESPONDING	EGI	POINTER	SUB-QUEUE-3
COUNT	ENABLE	POSITION	SYMBOLIC
DATE	END-OF-PAGE	QUEUE	TAPE
DAY	EOP	RECEIVE	TERMINAL
DEBUG-CONTENTS	EMI	REMOVAL	TEXT
DEBUG-ITEM	ESI	RENAMES	TIME
DEBUG-LINE	FOOTING	RESERVE	TOP
DEBUG-NAME	IN	RETURN	UNSTRING
DEBUG-SUB-1	KEPT	REVERSED	
DEBUG-SUB-2	LENGTH	REWIND	

Appendix B. CHARACTER SETS AND COLLATING SEQUENCE

ASCII	HEX	COBOL	ASCII	HEX	COBOL	ASCII	HEX	COBOL
NUL	00	x	/	2F			5E	x
SOH	01	x	0	30			5F	x
STX	02	x	1	31			60	x
ETX	03	x	2	32		a	61	
EOT	04	x	3	33		b	62	
ENQ	05	x	4	34		c	63	
ACK	06	x	5	35		d	64	
BEL	07	x	6	36		e	65	
BS	08	x	7	37		f	66	
HT	09	x	8	38		g	67	
LF	0A	x	9	39		h	68	
VT	0B	x	:	3A	x	i	69	
FF	0C	x	;	3B		j	6A	
CR	0D	x	<	3C		k	6B	
SO	0E	x	=	3D		l	6C	
SI	0F	x	>	3E		m	6D	
DLE	10	x	?	3F	x	n	6E	
DC1	11	x	@	40	x	o	6F	
DC2	12	x	A	41		p	70	
DC3	13	x	B	42		q	71	
DC4	14	x	C	43		r	72	
NAK	15	x	D	44		s	73	
SYN	16	x	E	45		t	74	
ETS	17	x	F	46		u	75	
CAN	18	x	G	47		v	76	
EM	19	x	H	48		w	77	
SUB	1A	x	I	49		x	78	
ESC	1B	x	J	4A		y	79	
FS	1C	x	K	4B		z	7A	
GS	1D	x	L	4C			7B	x
RS	1E	x	M	4D			7C	x
US	1F	x	N	4E			7D	x
space	20		O	4F			7E	x
!	21	x	P	50		DEL	7F	x
"	22		Q	51				
#	23	x	R	52				
$	24		S	53				

ASCII	HEX	COBOL	ASCII	HEX	COBOL	ASCII	HEX	COBOL
%	25	x	T	54				
&	26	x	U	55				
'	27	x	V	56				
(	28		W	57				
)	29		X	58				
*	2A		Y	59				
+	2B		Z	5A				
,	2C			5B	x			
-	2D			5C	x			
.	2E			5D	x			

Appendix C. GLOSSARY

INTRODUCTION

The terms in this Chapter are defined in accordance with their meaning as used in this document describing CIS 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 syntactical rules.

DEFINITIONS

Access Mode.
The manner in which records are to be operated upon within a file

Actual Decimal Point.
The physical representation, using either of the decimal point characters . (period) or , (comma) of the decimal point position in a data item.

Alphabet-Name.
A user-defined word in the SPECIAL-NAMES paragraph of the Environment Division that assigns a name to a specific character set and/or collating sequence.

Alphabetic Character.
A character that belongs to the following set. of letters: A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z and the space. Also a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,x,y and z which are converted to their upper case equivalents.

Alphanumeric Character.
Any character in the computer's character set.

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

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 two 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.

Called Program. A program which is the object of a CALL statement combined at run time with the calling program to produce a run unit.

Calling Program. A program which executes a CALL to another program.

Character. The basic indivisible unit of the language.

Character Set (CIS COBOL). The complete CIS COBOL character set consists of all characters listed below:

Character	**Meaning**
0,1, ... ,9	Numeric digit
A,B ... ,Z	Uppercase alphabetic
a,b ... ,z	Lowercase alphabetic
	Space (Blank)
+	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. Further characteristics of the physical storage are defined by the implementor.

Character-String. A sequence of contiguous characters which form a CIS 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 CIS COBOL character-strings whose purpose is to specify an attribute of an entry.

Collating Sequence. The sequence in which the characters that are acceptable in a computer are ordered for purposes of sorting, merging and or comparing.

Column.	A character position within a print line. The columns are numbered from one, by one, starting at the left-most character position of the print line and extending to the right-most character position of the print line.
Comment Entry.	An entry in the Identification Division that may be any combination of characters from the computer character set.
Comment Line.	A source program line represented by an asterisk in the indicator area of the line and any characters from the computer's character set in area A and area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a stroke (/) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line causes page ejection before printing the comment.
Compile Time.	The time at which an CIS COBOL source program is translated by the compiler to an CIS 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.
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.
Condition-Name.	The user-defined word assigned to a status of an implementor-defined switch or device.
Conditional Expression.	A simple condition specified in an IF, or PERFORM. (See Simple Condition and Complex Condition.)
Conditional Statement.	A conditional statement specifies that the truth value of a condition is to be determined, and that the subsequent action of the run-time program is dependent on this truth value.
Configuration Section.	A section of the Environment Division that describes overall specifications of source and run computers.
Connective.	A reserved word that is used to: 1. Associate a data-name, paragraph-name, condition-name, or text-name with its qualifier. 2. Link two or more operands written in a series. 3. Form conditions (logical connectives). (See Logical Operator.)
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 a keyboard by which an Operator can enter and receive visual data.
Currency Sign.	The character “$” (dollar sign) in the CIS 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 a CIS 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.
Cursor.	The indicator on a CRT screen that marks the line and character position which the input/output control is currently referencing.
Data Clause.	A clause that appears in a data description entry in the Data Division and provides information describing a particular attribute of a data item.
Data Description Entry .	An entry in the Data Division that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses as required.
Data Dictionary.	A dictionary file of user def in ed names constructed by the Compiler containing the number of bytes for each entry.
Data Item.	A character or set of contiguous characters (excluding in either case literals) defined as a unit of data by the CIS COBOL program.
Data-name.	A user-defined word that names a data item described in a data description entry in the Data Division. When used in the general formats , “data-name” represents a word which can neither be subscripted, nor indexed unless specifically permitted by the rules for that format.
Debugging Line.	A debugging line is any line with “D” in the indicator area of the line.
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 copy 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.

Digit Position.

A digit position is the amount of physical storage required to store a single digit. This amount varies depending on the usage of the data item describing the digit position. Further characteristics of the physical storage are defined by the implementor.

Division.

A set of sections or paragraphs (0 or more) that are formed and combined in accordance with a specific set of rules is called a division body. There are four divisions in a CIS COBOL program: Identification, Environment, Data and Procedure.

Division Header.

A combination of words followed by a period and a space that indicate the beginning of a division. The division headers are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION    USING data-name-1  data-n
```

Dynamic Access.

An access mode in which specific logical records can be obtained from or placed into a disk file in a non-sequential manner (see Random Access) and obtained from a file in a sequential manner (see Sequential Access) during the scope of the same OPEN statement.

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 CIS COBOL source program after which no further procedures appear. Any descriptive set of consecutive clauses terminated by a period (.) and written in the Identification Division, Environment Division or Data Division of an CIS COBOL source program.
Environment Clause.	A clause that appears as part of an Environment Division entry.
Extend Mode.	With the EXTEND phrase specified, the state of a file after execution of an OPEN statement, and before the execution of a CLOSE statement for the file.
Figurative Constant.	A compiler-generated value referenced through the use of certain reserved words.
File.	A collection of records.
File Clause.	A clause that appears as part of any of the following Data Division entries: File Description (FD)
FILE-CONTROL.	The name of an Environment Division paragraph in which the data files for a given source program are declared.
File Description Entry.	An entry in the File Section of the Data Division that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.
File-Name.	A user-defined word that names a file described in a file description entry or a sort-merge file description entry within the File Section of the Data Division.
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 Program.	A screen formatting program that automatically generates CIS 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.
Identifier.	A data-name, followed as required by the syntactically correct combination of subscripts and indices necessary to make unique reference to a data item.

Imperative Statement.	A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements.
Implementor-Name.	A system-name that refers to a particular feature avail- able on the implementors computing system.
Index.	A computer storage position or register, the contents of which represent the identification of a particular element in a table.
Index Data Item.	A data item in which the value associated with an index-name can be stored in a form specified by the implementor.
Indexed File.	A file with indexed organization.
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 CIS 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.
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 the term '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.
Intermediate Code	The code produced by the CIS COBOL compiler from the source code entered, and which the Run Time System 'fast loads' for execution.
Invalid Key Condition.	A condition, at object time, caused when a specified value of the key associated with an indexed or relative file is determined to be invalid.
Issue Disk.	The flexible diskette or which the CIS COBOL software is supplied to users.
Key.	A data item which identifies the location of a record, or a set of data items which serve to identify the ordering of data.
Key of Reference.	The key currently being used to access records within an indexed file.
Key Word.	A reserved word whose presence is required when the format in which the word appears is used in a source program.
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-number 77 identifies special properties of a data description entry.
Library-Name.	A user-defined word that names a CIS 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 C/R (carriage return) and L/F (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.
Logical Operator.	The reserved word 'NOT'. It can be used for logical negation.
Logical Record.	The most inclusive data item. The level-number for a record is 01.
Low Order End.	The rightmost character of a string of characters.
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 Simple Condition.	The 'NOT' logical operator immediately followed by a simple condition.
Next Executable Sentence.	The next sentence to which control will be transferred after execution of the current statement is complete.
Next Executable Statement.	The next statement to which control will be transferred after execution of the current statement is complete.
Next Record.	The record which logically follows the current record of a file.
Non contiguous Items.	Elementary data items, in the Working-Storage and Linkage Sections, which bear no hierarchic relationship to other data items.
Nonnumeric Item.	A data item whose description permits its contents to be composed of any combination of characters taken from the

	computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.
Nonnumeric Literal.	A character-string bounded by quotation marks. The string of characters may include any character in the computer's character set. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used.
Numeric Character.	A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
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.
Output File.	A file that is opened in either the output mode or extend mode.
Output-Mode.	The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified for that file and before the execution of a CLOSE statement for that file.
Paragraph.	In the Procedure Division, a paragraph-name followed by a period and a space and optionally by one, or more sentences. In the Identification and Environment Divisions, a paragraph header followed by zero, one, or more entries.
Paragraph Header.	A reserved word, followed by a period and a space that indicates the beginning of a paragraph in the Identification and Environment Divisions. The permissible paragraph headers are:

In the Identification Division:

```
PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

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 CIS COBOL procedural statement or of a COBOL clause.

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.

Punctuation Character. A character that belongs to the following set:

Character	**Meaning**
,	comma
;	semicolon
.	period
"	quotation mark
(	left parenthesis
)	right parenthesis
	space
=	equal sign

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.

Reference-Format.

A format that provides a standard method for describing COBOL source programs.

Relation.

(See Relational Operator)

Relation Character.

A character that belongs to the following set:

Character	Meaning
>	greater than
<	less than
=	equal to

Relation Condition.

The proposition, for which a truth value can be determined, that the value of an arithmetic expression or data item has a specified relationship to the value of another arithmetic expression or data item. (See Relational Operator),

Relational Operator.

A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meaning are:

Relational Operator	Meaning
IS NOT GREATER THAN IS NOT >	Greater than or not greater than
IS NOT LESS THAN IS NOT <	Less than or not less than
IS NOT EQUAL THAN IS NOT =	Equal to or not equal to

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 Debug. An option available to CIS COBOL programmers entered as a user option enabling break-point facilities in run time programs.

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 CIS COBOL compiler and enables it to be executed by providing interfaces to the operating system and CRT.

Run Unit. A set of one or more intermediate code programs which function, at run time, as a unit to provide problem solutions.

Section. A set of none, one, or more paragraphs or entries, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

Section Header. A combination of words followed by a period and a space that indicates the beginning of a section in the Environment, Data and Procedure Division.

In the Environment and Data Divisions, a section header is composed of reserved words followed by a period and a space. The permissible section headers are:

In the Environment Division:

```
CONFIGURATION SECTION
INPUT-OUTPUT SECTION
```

In the Data Division:

```
FILE SECTION
WORKING-STORAGE SECTION
LINKAGE SECTION
```

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a segment-number (optional), followed by a period and a space.

Section-Name. A user-defined word which names a section in the Procedure Division.

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.

Simple Condition. Any single condition chosen from the set:

relation condition
class condition
switch-status condition
sign condition
(simple-condition)

SOURCE-COMPUTER. The name of an Environment Division paragraph in which the computer environment, within which the source program is compiled, is described.

Source Program. Although it is recognised that a source program may be represented by other forms and symbols, in this document it always refers to a syntactically correct set of COBOL statements beginning with an Identification Division and ending with the end of the Procedure Division. In contexts where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'source program'.

Special Character. A character that belongs to the following set:

Character	**Meaning**
+	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

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.
Standard Data Format.	The concept used in describing the characteristics of data in a COBOL Data Division under which the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.
Statement.	A syntactically valid combination of words and symbols written in the Procedure Division beginning with a verb.
Subprogram.	(See Called Program).
Subscript.	An integer whose value identifies a particular element in a table.
Subscripted Data-Name.	An identifier that is composed of a data-name followed by one or more subscripts enclosed in parenthesis.
Switch-Status Condition.	The proposition, for which a truth value can be determined, that an implementor-defined switch, capable of being set to an 'on' or 'off' status, has been set to a specified status.
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.
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.
Unary Operator.	A plus (+) or a minus (-) sign, which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression of +1 or -1 respectively.
User-Defined Word.	A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.
Variable.	A data item whose value may be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

Verb.	A word that expresses an action to be taken by a COBOL compiler or run time program.
Word.	A character-string of not more than 30 characters which forms a user-defined word, a system-name, or a reserved word.
Working-Storage Section.	The section of the Data Division that describes working storage data items, composed either of noncontiguous items or of working storage records or of both.
77 Level-Description-Entry.	A data description entry that describes a noncontiguous data item with the level-number 77.

Appendix D. COMPILE-TIME ERRORS

The error descriptions that correspond to error numbers as printed on listings produced by the CIS COBOL compiler are as follows:

ERROR	DESCRIPTION
01	Compiler Error; consult your Technical Support Service
02	Illegal format of data-name
03	Illegal format of literal or invalid use of 'ALL'
04	Illegal format of character
05	Data-name declared twice
06	Too many data or procedure names have been declared - compilation abandoned
07	Illegal character in column 7, or continuation line error
08	Nested COPY statement or unknown file specified
09	'.' missing
10	The statement starts in the wrong area of the source line
22	'DIVISION' missing
23	'SECTION' missing
24	'IDENTIFICATION' missing
25	'PROGRAM-ID' missing
26	'AUTHOR' missing
27	'INSTALLATION' missing
28	'DATE-WRITTEN' missing
29	'SECURITY' missing
30	'ENVIRONMENT' missing
31	'CONFIGURATION' missing
32	'SOURCE-COMPUTER' missing
33	OBJECT-COMPUTER or SPECIAL-NAMES clause in error
34	'OBJECT-COMPUTER' missing
36	'SPECIAL-NAMES' missing
37	SWITCH Clause in error
38	DECIMAL-POINT Clause in error
39	CONSOLE Clause in error
40	Illegal currency symbol
42	'DIVISION' missing
43	'SECTION' missing
44	'INPUT-OUTPUT' missing
45	'FILE-CONTROL' missing
46	'ASSIGN' missing
47	'SEQUENTIAL' or 'RELATIVE' or 'INDEXED' missing
48	'ACCESS' missing on indexed or relative file
49	'SEQUENTIAL' or 'DYNAMIC' missing
50	Illegal combination ORGANIZATION/ACCESS/KEY

ERROR	DESCRIPTION
51	Unrecognised clause in SELECT statement
52	RERUN clause contains syntax error
53	SAME AREA clause contains syntax error
54	File-name missing or illegal
55	'DATA DIVISION' missing
56	'PROCEDURE DIVISION' missing or unknown statement
57 *	'EXCLUSIVE', 'AUTOMATIC' or 'MANUAL' missing
58 *	Non-exclusive lock mode specified for restricted file
62	'DIVISION' missing
63	'SECTION' missing
64	File-name not specified in SELECT statement
65	RECORD SIZE integer missing
66	Illegal level number or level 01 required
67	FD qualification contains syntax error
68	'WORKING-STORAGE' missing
69	'PROCEDURE DIVISION' missing or unknown statement
70	Unrecognized clause in Data Description or previous'.' missing
71	Incompatible clauses in Data Description
72	BLANK is illegal with non-numeric data-item
73	PICTURE clause too long
74	VALUE with non-elementary item, wrong data-type or value truncated
75	VALUE clause in error or illegal for PICTURE type
76	FILLER/SYNCHRONIZED/JUSTIFIED/BLANK clause for non-elementary item
77	Preceding item at this level has 0 or more than 8192 bytes
78	REDEFINES of different levels or unequal field lengths.
79	Data Division exceeds 32K and data-item has address above 7FFF
81	Data Description clause inappropriate or repeated
82	REDEFINES data-name not declared
83	USAGE must be COMP, DISPLAY or INDEX
84	SIGN must be LEADING or TRAILING
85	SYNCHRONIZED must be LEFT or RIGHT
86	JUSTIFIED must be RIGHT
87	BLANK must be ZERO
88	OCCURS must be numeric, non-zero and unsigned
89	VALUE must be a literal, numeric literal or figurative constant
90	PICTURE string has illegal precedence or illegal character
91	INDEXED data-name missing or already declared
92	Numeric edited PICTURE string is too large
101	Unrecognised verb
102	IF ELSE mismatch
103	Data-item has wrong data-type or is not declared

ERROR	DESCRIPTION
104	Procedure name has been declared twice
10S	Procedure name is the same as a data-name
106	Name required
107	Wrong combination of data-types
108	Conditional statement not allowed; imperative statement expected
109	Malformed subscript
110	ACCEPT or DISPLAY wrong
111	Illegal syntax used with I-O verb
112 *	LOCK clause specified for file with lock mode EXCLUSIVE
113 *	KEPT specified for uncommittable file
115 *	KEPT omitted for comittable file
116	IF statements nested too deep (maximum 8)
117	Structure of Procedure Division wrong (e.g. DECLARATIVES not first)
118	Reserved Word missing or incorrectly used
119	Too many subscripts in one statement
120	Too many operands in one statement
141	Inter-segment procedure name declared twice
142	IF ELSE mismatch at the end of source input
143	Data-Item has wrong data-type or is not declared
144	Procedure name undeclared
145	INDEX name declared twice
146	Cursor address field not declared or not 4 bytes long
147	KEY declaration missing or FD missing
148	STATUS declaration missing
149	FILE STATUS data-item has the wrong format
150	Paragraph to be ALTERed is not declared
151	PROCEDURE DIVISION in error
152	USING parameter is not declared in the linkage section
153	USING parameter is not level 01 or 77
154	USING parameter is used twice in the parameter list
157	Structure of Procedure Division wrong (e.g. DECLARATIVES not first)
160	Too many operands in one statement

* The error codes marked by an asterisk apply only when the optional FILESHARE product is in use.

In addition to these numbered error messages, the following message can be displayed with subsequent termination of the compilation:

```
FATAL I-O ERROR:  filename
```

where filename is the erroneous file.

Any intermediate code file produced is not usable.

The following conditions will cause this error:

Disk overflow
File directory overflow
File full
Impossible I-O device usage

Other operating system dependent conditions can also cause this error.

Note

You will notice that the numbers of the numbered error messages listed above are not continuous, i.e., there are gaps in the numbering. The compiler should never have cause to generate an error message with a number not listed above. If you ever encounter such a number, consult your Micro Focus Product Technical Support office.

Appendix E. RUN-TIME ERRORS

Run-time error messages are preceded by the name and segment number of the currently executing intermediate code file.

There are two types of runtime errors: Recoverable and Fatal.

(a) Recoverable errors

If the programmer has specified the STATUS clause in the FILE-CONTROL paragraph of a program error handling is the programmer's responsibility. This will generally only apply to errors that are not considered fatal by the operating system. (See File Status in Chapters 5, 6 and 7)

(b) Fatal errors

All errors except those above are fatal. They may come from the operating system or from the run-time system. Fatal errors cause a message to be output to the console which includes a 3-digit error code and reference to the COBOL statement subsequent to that in which the error occurred. These fall into two classes:

(i) Exceptions — These cover arithmetic overflow, subscript out of range, too many levels of perform nesting.

(ii) I-O errors — These exclude those for which STATUS is not selected as above.

Error	**Description**
151	Random read on sequential file
152	REWRITE on file not open I-O
153	Subscript out of range
154	Perform nesting exceeds 22 levels
156	Invalid file operation
157	Object file too large
158	REWRITE on line-sequential file
159	Malformed line-sequential file
161	Illegal intermediate code
162	Arithmetic overflow or underflow
164	Specified CALL code not supplied or Attempt to call a COBOL module recursively (i.e when is already active)
165	Incompatible releases of compiler and run-time system
168	Memory arrangement failure
169	Invalid indirect sequential file key length (>32 characters)
170	Illegal operation in Indexed Sequential
171	Attempt to read I-S record in output/extend mode
172	Attempt to delete I-S record in non I-O mode
173	Attempt to write I-S record in input mode
174	Attempt to CALL/CANCEL on active program
176	Illegal inter-segment reference
180	COBOL file malformed
181	Fatal file malformation

Error	Description
194 (CP/M 1.4 only)	File size too large (>0.5MB) or Failure to Open on Extent
195	DELETE/REWRTTE not preceded by a READ
196	Relative (or Indexed) - Record number too large (>65535)
197	File save failure
198	Program load failure (using CHAIN)
199	Indexed sequential file name too long (>20 characters)
200	Insufficient space to load Animator

See also appendix D in the *CIS COBOL Operating Guide* specific to your operating system.

Appendix F. SYNTAX SUMMARY

All the syntax for CIS COBOL is summarized below.

E denotes that the feature is a CIS COBOL extension to ANSI COBOL.

D denotes that the feature is documentary only in CIS COBOL.

GENERAL FORMAT FOR IDENTIFICATION DIVISION

[IDENTIFICATION DIVISION.]
[PROGRAM-ID.] program name
[AUTHOR.] [comment entry]...
[INSTALLATION.] [comment entry]...
[DATE-WRITTEN.] [comment entry]...
[DATE-COMPILED.] [comment entry]...
[SECURITY.] [comment entry]...

GENERAL FORMAT FOR ENVIRONMENT DIVISION

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. source-computer-entry [WITH DEBUGGING MODE] .
OBJECT-COMPUTER. object-computer-entry
[,MEMORY SIZE *integer* { WORDS | CHARACTERS | MODULES }]
[,PROGRAM COLLATING SEQUENCE IS *alphabet-name*]
SPECIAL-NAMES.
SWITCH {0 ... 7} [IS mnemonic-name] { ,ON STATUS IS condition-name-1 | [,OFF STATUS IS condition-name-2] | ,OFF STATUS IS condition-name-2 | [,ON STATUS IS condition-name-1]}
[{ ,SYSIN | ,SYSOUT } IS mnemonic-name]
[, TAB IS mnemonic-name]
[, CURRENCY SIGN IS literal-9]
[, DECIMAL-POINT IS COMMA]
[, CONSOLE IS CRT]
[, CURSOR IS data-name-1] .
INPUT-OUTPUT SECTION.
FILE-CONTROL.
file-control-entry ...
I-O-CONTROL. [; RERUN [ON { file-name-1 | implementor-name }] EVERY {{ [END OF] { REEL | UNIT } | integer-1 RECORDS } | OF file-name-2 | integer-2 CLOCK-UNITS | condition-name }]...
[; SAME AREA FOR file-name-3 [, file-name-4]...]... .

GENERAL FORMAT FOR FILE-CONTROL ENTRY

Sequential SELECT:

SELECT file-name ASSIGN TO { external-file-name-literal | file-identifier } [, external-file-name-literal | file-identifier]

[; ORGANIZATION IS SEQUENTIAL | LINE SEQUENTIAL]
[; ACCESS MODE IS SEQUENTIAL]
[; FILE STATUS IS data-name-1]

Relative Select:

SELECT file-name
ASSIGN TO { external-file-name-literal | file-identifier } [, { external-file-name-literal | file-identifier }]
; ORGANIZATION IS RELATIVE
[; ACCESS MODE IS { SEQUENTIAL ,RELATIVE KEY IS data-name | { RANDOM | DYNAMIC } ,RELATIVE KEY IS data-name }]
[; FILE STATUS IS data-name-2]

Indexed Select:

SELECT file-name
ASSIGN TO { external-file-name-literal | file-identifier } [, { external-file-name-literal | file-identifier }]
; ORGANIZATION IS INDEXED
[; ACCESS MODE IS { SEQUENTIAL | RANDOM | DYNAMIC }]
; RECORD KEY IS data-name-1
[; FILE STATUS IS data-name-3]

GENERAL FORMAT FOR THE DATA DIVISION

DATA DIVISION.
[FILE SECTION.
FD file-name
[; BLOCK CONTAINS integer { RECORDS | CHARACTERS }]
[; RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS]
; LABEL { RECORD IS | RECORDS ARE } { STANDARD | OMITTED }
; VALUE OF data-name-1 IS literal-1 [, data-name-2 IS literal-2]...
; DATA { RECORD IS | RECORDS ARE } data-name-1 [, data-name-2]...
[; CODE-SET IS alphabet-name] .
[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 }]...]

GENERAL FORMAT FOR DATA DESCRIPTION ENTRY

level-number { data-name-1 | FILLER }
[; REDEFINES data-name-2]
[{ PICTURE | PIC } IS character-string]
[; [USAGE IS] { COMPUTATIONAL | COMP | COMPUTATIONAL-3 | COMP-3 | DISPLAY }]
[; [SIGN IS] { LEADING | TRAILING } [SEPARATE CHARACTER]]
[; { SYNCHRONIZED | SYNC } { LEFT | RIGHT }]
[; { JUSTIFIED | JUST } RIGHT] [; BLANK WHEN ZERO]
[; VALUE IS literal]

GENERAL FORMAT FOR PROCEDURE DIVISION

Declarative format:

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

[DECLARATIVES. { section-name SECTION [segment-number] . declarative-sentence
[paragraph-name. [sentence]...]... }...
END DECLARATIVES.]
{ [section-name SECTION [segment-number]]
[[paragraph-name] [sentence]...] }

Non-declarative format:

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

GENERAL FORMAT FOR VERBS

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

ACCEPT identifier [FROM CONSOLE]

ADD { identifier-1 | literal-1 } [, { identifier-2 | literal-2 }]... TO identifier-m [ROUNDED]
[, identifier-n[ROUNDED]]... [; ON SIZE ERROR imperative-statement]

ADD { identifier-1 | literal-1 } , { identifier-2 | literal-2 } [, { identifier-3 | literal-3 }]... GIVING identifier-m [ROUNDED] [, identifier-n [ROUNDED]] [; ON SIZE ERROR imperative-statement]

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

CALL { identifier-1 | literal-1 } [USING data-name-1 [, data-name-2]...] [ON OVERFLOW imperative-statement]

CANCEL { identifier-1 | literal-1 } [{ identifier-2 | literal-2 }]...

DELETE file-name RECORD [;INVALID KEY imperative-statement]

DISPLAY { identifier-1 | literal-1 } [, { identifier-2 | literal-2 }]... [UPON CONSOLE]

DISPLAY { data-name-1 | literal-3 } [AT { data-name-2 | literal-4 }] UPON { CRT | CRT-UNDER }

DIVIDE { identifier-1 | literal-1 } INTO identifier-2 [ROUNDED]
[, identifier-3 [ROUNDED]]... [; ON SIZE ERROR imperative-statement]

DIVIDE { identifier-1 | literal-1 } INTO { identifier-2 | literal-2 } GIVING identifier-3 [ROUNDED]
[, identifier-4 [ROUNDED]]... [; ON SIZE ERROR imperative-statement]

DIVIDE { identifier-1 | literal-1 } BY { identifier-2 | literal-2 } GIVING identifier-3 [ROUNDED]
[, identifier-4 [ROUNDED]]... [; ON SIZE ERROR imperative-statement]

ENTER language-name [routine-name]

EXIT [PROGRAM]

GO TO {procedure-name-1}

GO TO procedure-name-1 [, procedure-name-2]... , procedure-name-n DEPENDING ON identifier

IF condition; [THEN] { statement-1 | NEXT SENTENCE } { ; ELSE statement-2 | ; ELSE NEXT SENTENCE }

INSPECT identifier-1 TALLYING identifier-2 FOR , { ALL | LEADING | CHARACTERS } { identifier-3 | literal-1 } [{ BEFORE | AFTER } INITIAL { identifier-7 | literal-5 }]

INSPECT identifier-1 REPLACING
{ CHARACTERS BY identifier-6 | literal-4 | , { ALL | LEADING | FIRST } , { identifier-5 | literal-3 } BY { identifier-6 | literal-4 } }
[{ BEFORE | AFTER } INITIAL { identifier-7 | literal-5 }]

INSPECT identifier TALLYING tally-clause REPLACING replacing-clause

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

MULTIPLY { identifier-1 | literal-1 } BY identifier-2 [ROUNDED]
[, identifier-3 [ROUNDED]]... [; ON SIZE ERROR imperative-statement]

MULTIPLY { identifier-1 | literal-1 } BY { identifier-2 | literal-2 } GIVING identifier-3 [ROUNDED]
[, identifier-4 [ROUNDED]]... [; ON SIZE ERROR imperative-statement]

OPEN { INPUT file-name-1 [, file-name-2]... OUTPUT file-name-3 [, file-name-4]... I-O file-name-5 [, file-name-6]... EXTEND file-name-7 [, file-name-8]... }

PERFORM procedure-name-1 [{ THROUGH | THRU } procedure-name-2]

PERFORM procedure-name-1 [{ THROUGH | THRU } procedure-name-2] { identifier-1 | integer-1 } TIMES

PERFORM procedure-name-1 [{ THROUGH | THRU } procedure-name-2] UNTIL condition-1

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

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

REWRITE record-name [FROM identifier] [; INVALID KEY imperative-statement]

SET { identifier-1 [, identifier-2]... | index-name-1 [, index-name-2]... } { TO | UP BY | DOWN BY } { identifier-3 | index-name-3 | integer-1 }

START file-name [KEY IS = | IS > | IS NOT < data-name
[;INVALID KEY imperative-statement]]

STOP { RUN | literal }

SUBTRACT { identifier-1 | literal-1 } , { identifier-2 | literal-2 }... ... FROM identifier-m [ROUNDED]
[, identifier-n [ROUNDED]]... [; ON SIZE ERROR imperative-statement]

SUBTRACT { identifier-1 | literal-1 } , { identifier-2 | literal-2 }... ... FROM identifier-m GIVING identifier-n [ROUNDED] [, identifier-o [ROUNDED]]... [; ON SIZE ERROR imperative-statement]

USE AFTER STANDARD { EXCEPTION | ERROR } PROCEDURE ON { file-name-1 | INPUT | OUTPUT | I-O | EXTEND }

USE FOR DEBUGGING ON { procedure-name-1 | ALL PROCEDURES } [, { procedure-name-2 | ALL PROCEDURES }...] OUTPUT

WRITE record-name [FROM identifier-1] [BEFORE | AFTER ADVANCING integer LINE | LINES | TAB | PAGE]

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

GENERAL FORM FOR COPY STATEMENT

```
COPY text-name | external-file-name-literal .
```

Appendix G. SUMMARY OF EXTENSIONS TO ANSI COBOL

CIS COBOL is oriented to microcomputer users with the system readily accessible and usually with a CRT. CIS COBOL therefore provides extensions for interactive working, program control of files, text file handling and rapid development and testing. These facilities are summarized below.

SCREEN FORMATTING AND DATA ENTRY

THE ACCEPT STATEMENT

An additional format for the ACCEPT statement is provided as follows:

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

data-name-2	allows the start of screen to be changed dynamically. It refers to a PIC 9999 field where the most significant 99 is a line count 1-25 and the least significant 99 is a character position 1-80.
data-name-1	refers to a record, group or elementary item but may not be subscripted.
literal-1	is a numeric literal

NOTE: See Chapter 3 for description. See also Appendix H for Environment Division changes.

THE DISPLAY STATEMENT

An additional format for the DISPLAY statement is provided as follows:

Format

```
DISPLAY { data-name-1 | literal-3 } [AT { data-name-2 | literal-1 }] UPON { CRT | CRT-UNDER }
```

literal-3	is an alphanumeric literal
dataname-1	refers to a record, group or elementary item but may not be subscripted
dataname-2	defines the left-most position on the screen. It refers to a PIC 9999 field where the most significant 99 is a line count 1-25 and the least significant 99 is a character position 1-80.

NOTE: See Chapter 3 for description.

DISK FILES

Two extensions are offered by CIS COBOL file processing; these are as follows:

1. Line sequential files
2. Run time input of filenames

LINE SEQUENTIAL FILES

When LINE SEQUENTIAL ORGANIZATION is specified in the FILE CONTROL paragraph ORGANIZATION IS entry, the file is treated as consisting of variable length records separated by the lire delimiter characters. Trailing spaces in output records are replaced by a record terminator which is operating system dependent.

RUN TIME INPUT OF FILENAMES

The ASSIGNed name in the SELECT statement for a file is processed on OPENing as follows:

When the INPUT or OUTPUT phrase is specified, execution of OPEN causes checking of the file names in accordance with the operating system conventions for opening an input or output file. The full operating system features for file reallocation and device control are therefore available to the CIS COBOL program.

LOWER CASE CHARACTERS

The full alphanumeric lower case a to z is available in CIS COBOL. Reserved and user word characters are read as their upper case equivalents (A to Z).

HEXADECIMAL VALUES

Hexadecimal binary values can be attributed to non-numeric literals in CIS COBOL by expressing them as X "xx", where x is a hexadecimal character in the set 0-9, A-F; xx can be repeated up to 120 times, but the number of hexadecimal digits must be even.

INTERACTIVE DEBUGGING

There is a Run-Time Debug Package to provide break-point facilities in the user's program. Programs may be run from the start until a specified break-point is reached, when control is passed back to the user. At this point, data areas may be inspected or changed.

The Debug package is entered as an option by the user and the user program is then tested line by line, paragraph by paragraph and so on as required. The commands to the package can reference procedure statements and data areas by means of a 4-digit hexadecimal code output by the compiler against each line of the compilation listing. Powerful macros of commands can be used to give very sophisticated debugging facilities. The precise details for using the package vary according to the host operating system and are described in the appropriate Operating Guide.

Appendix H. SYSTEM DEPENDENT LANGUAGE FEATURES

This Appendix summarizes those parts of a COBOL program that need to be changed to run them as CIS COBOL programs and those parts that do not need changing specifically but are ignored by the CIS COBOL compiler when generating the object program.

MANDATORY CHANGES

ENVIRONMENT DIVISION

The only statements in the environment division that must be specialized for CIS COBOL are shown below:

Configuration Section

```
SPECIAL-NAMES. special names entry
```

special names entry must include the following:

```
CURSOR IS data-name-1
```

The CURSOR IS data-name-1 clause specifies the data-name which will contain the CRT cursor address as used by ACCEPT statements. Data-name-1 must be declared in the Working-Storage section as a 4 character item. The interpretation of the 4 characters is given in the ACCEPT statement description.

Input-Output Section

File names must be as described in Appendix F of the *CIS COBOL Operating Guide*.

STATEMENTS COMPILED AS DOCUMENTATION ONLY

COBOL programs not specifically written for compilation as CIS COBOL on microcomputers can still be compiled. Statements using features that are not available are treated as documentary only, and are not compiled. A summary of these features follows:

ENVIRONMENT DIVISION

I-O-Control Paragraph

The clauses that refer to a real time clock and magnetic tape in this paragraph are ignored by the compiler during compilation but do not cause compile times errors. These clauses are as follows:

END OF { REEL | UNIT } of file-name-2

(no magnetic tape)

integer-2 CLOCK UNITS

(no clock)

DATA DIVISION

File Description Paragraph

The following complete statements in the file description are ignored by the compiler during compilation but do not cause compile time errors:

BLOCK CONTAINS integer-1 TO integer-2 { RECORDS | CHARACTERS }

CODE-SET IS alphabetic-name

LABEL { RECORD IS | RECORDS ARE } { STANDARD | OMITTED }

VALUE OF implementor-name-1 IS literal-1 [, implementor-name-2 IS literal-2]

PROCEDURE DIVISION

CLOSE Statement

The following phrases in the CLOSE statement are ignored by the compiler during compilation but do not cause compiler-time errors:

{ REEL | UNIT }

(No magnetic tape)

Appendix I. LANGUAGE SPECIFICATION

CIS COBOL is ANSI COBOL as given in "American National Standard Programming Language COBOL" (ANSI X.3.23 1974). CIS COBOL implements both levels of ANSI COBOL. The following modules are fully implemented at Level 1:

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

In addition many Level 2 features are implemented such as:

- Nucleus - Nested IF, PERFORM UNTIL
- Relative and Indexed sequential I/O - START statement
- Inter-Program Communication - Fully implemented at level 2

This appendix specifies the implementation of Version 4.3 CIS COBOL. The implementation of each of the eight standard COBOL modules listed above is given under the following headings as applicable:

Level 1 Implementation
Level 2 Implementation
CIS COBOL Extensions

Appendix F in this manual is a CIS COBOL syntax summary.

NUCLEUS

Level One Implementation

Fully implemented to Level One.

Level Two Implementation

1. DATE-COMPILED in the Identification Division is accepted for documentation purposes only.
2. Up to 49 Level Numbers are permitted and 1-9 can be a single digit.
3. The characters , and ; are permitted as separators

4. The character '>', '=' and '<' are permitted in relative conditions.
5. The PERFORM ... THROUGH ... UNTIL feature is implemented.
6. Plural forms of the figurative constants can be used.
7. IF statements can be nested.
8. Mnemonic names are permitted in ACCEPT and DISPLAY statements (See CIS COBOL extensions 6 and 7 below).
9. Procedure names can be all digits.
10. REDEFINES clauses can be nested.
11. Non-numeric operands can be compared.

CIS COBOL Extensions

1. Lower case letters a to z are read as upper case letters A to Z.
2. Hexadecimal binary values can be attributed to non-numeric values by expressing literals as X"nn".
3. Reserved word SPACE can be used to clear the whole CRT screen.
4. ANS switch not set enables omission of certain ANSI required "red tape" paragraphs and statements.
5. COMPUTATIONAL-3 or COMP-3 can be specified in the USAGE clause to specify packed internal decimal storage, (BCD).
6. `ACCEPT` data-name-1 [AT { data-name-2 | literal-1 }] FROM CRT

   gives enhanced CRT input features
7. `DISPLAY` { data-name-1 | literal-1 } [AT { data-name-2 | literal-2 }] UPON { CRT | CRT-UNDER }

   gives enhanced CRT output facilities.
8. 'CURSOR IS data-name' can be specified in SPECIAL-NAMES and 'data-name' in WORKING-STORAGE section to specify CRT cursor address for ACCEPT statements.

SEQUENTIAL, RELATIVE AND INDEXED I-O

Level One Implementation

Fully implemented to Level One.

Level Two Implementation

1. The START statement is fully supported for Relative and Indexed files.
2. In sequential files, EXTEND is supported.
3. In OPEN and CLOSE statements:

   REEL

UNIT

are accepted for documentation purposes only.

4. LOCK in the CLOSE statement is treated as documentary only.
5. Dynamic access mode and READ NEXT are supported for relative and indexed files.
6. Only the first assignment in each ASSIGN is actioned, others are treated as documentary only at compilation.
7. The I-O-CONTROL paragraph is treated as documentary only as are its RERUN and SAME AREA clauses.
8. The following are treated as documentary only in the FD clause:

   BLOCK CONTAINS
   CODE-SET
   DATA RECORDS
   LABEL RECORDS
   RECORDS CONTAINS
   VALUE OF

CIS COBOL Extensions

1. Run Time allocation of file-names. See Appendix F in the *CIS COBOL Operating Guide*.
2. LINE SEQUENTIAL is an additional file type.
3. All File Description (FD) clauses are optional when ANS switch is unset.
4. Tabbing is available, specified by TAB in the WRITE statement.

TABLE HANDLING

Level One Implementation

Fully implemented to Level One.

CIS COBOL Extensions

1. Items can be accessed in tables up to 49 dimensions. This extension is restricted to three dimensions if the ANS switch is set.

SEGMENTATION

Level One Implementation

Fully implemented to Level One.

LIBRARY

Level One Implementation

Fully implemented to Level One.

DEBUG

Level One Implementation

Fully implemented to Level 1 plus an additional Interactive Run-Time Debug package.

CIS COBOL Extensions

A powerful Run-Time Debug package is available. See Chapter 3 in the *CIS COBOL Operating Guide*.

INTER-PROGRAM COMMUNICATION

Level Two Implementation

Fully implemented to Level Two.

Index

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z

Colophon

This book was reconstructed into DocBook format from a scanned PDF found on the Internet. The PDF file already had OCR performed and the text was embedded in the file.

The original was published by Acorn Computers Limited in cooperation with the British Broadcasting Corporation.

Source version: 1.0.2