COBOL85

Reference Manual

DPS7000/XTA
NOVASCALE 7000

Languages: COBOL

REFERENCE
47 A2 05UL 04

DPS7000/XTA NOVASCALE 7000

COBOL85

Reference Manual

Languages: COBOL

November 1997

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
47 A2 05UL 04

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

To order additional copies of this book or other Bull Technical Publications, you are invited to use the Ordering Form also provided at the end of this book.

Trademarks and Acknowledgements

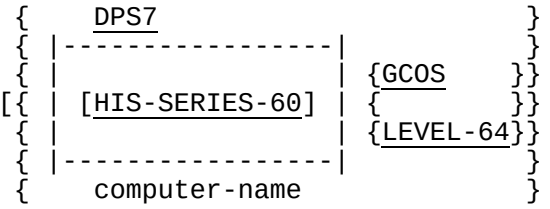
Preface

COBOL (Common Business Oriented Language), a programming language used for commercial data processing applications, was developed by the Conference on data Systems Languages (CODASYL) American National Standard COBOL x3.23.1985 as specified by the American National Standards Committee or the identical International Standards Organization (ISO) standard No. 1989, is the standard for the language implemented by the GCOS (General Comprehensive Operating Supervisor) DPS 7000 COBOL Compiler.

MANUAL OBJECTIVES

This manual is designed to provide a complete description of the DPS 7000 COBOL language as developed for use with the DPS 7000 Computer System.

|Underlined text delimited by vertical bars| or boxed elements in formats (as shown immediately below)



in this manual indicate DPS 7000 extensions to American National Standard COBOL. Use of these extensions in COBOL source programs intended to be compiled interchangeably by more than one COBOL compiler may lead to different compilation and/or execution results.

This COBOL manual has been developed as the basic reference source for all DPS 7000 COBOL language rules, including language structure, format and usage.

Only those features described in this manual should be used in the source program input to the DPS 7000 COBOL compiler. Any other COBOL statement may result in undefined output.

INTENDED AUDIENCE

Persons concerned with COBOL DPS 7000 programming:

- Programmers
- System engineers

STRUCTURE OF THE DOCUMENT

The first part (Chapter 1 through Chapter 4) of the manual covers general characteristics of the language in narrative form.

The second part (Chapter 5 through Chapter 13) of the manual consists of the specific format descriptions for all COBOL Statements, clauses, and phrases available in this release. Specific rules related to language or syntax requirements appear under each format description, followed by general rules for usage of the term in a program.

A third part (Chapters 14 through 18) gives description of segmentation, COBOL source text manipulation facility, debugging facility, COBOL reference format, and intrinsic functions.

ASSOCIATED DOCUMENTS

The following publications of the DPS 7000 manual set should also be referred to:

Program Checkout Facility User's Guide *47 A2 15UP*

COBOL 85 User's Guide *47 A2 06UL*

SORT/MERGE Utilities User's Guide *47 A2 08UF*

GCOS 7-V6 Networks: Overview and Generation *47 A2 71UC*
GCOS 7-V6 Networks: Operations Reference Manual *47 A2 72UC*
GCOS 7-V6 Networks: DSAC User's Guide *47 A2 75UC*
GCOS 7-V6 Networks: AUPI User's Guide *47 A2 76UC*

Networks Overview (V7) *47 A2 92UC*
Networks Generation (7) *47 A2 93UC*
Networks User's Guide (V7) *47 A2 94UC*

MCS User's Guide *47 A2 32UC*

JCL Reference Manual *47 A2 11UJ*
JCL User's Guide *47 A2 12UJ*

Library Maintenance Reference Manual *47 A2 01UP*
Library Maintenance User's Guide *47 A2 02UP*

GCOS 7-V6 Data Management Utilities User's Guide *47 A2 26UF*

TDS COBOL Programmer's Guide (V6) *47 A2 21UT*
TDS COBOL Programmer's Guide (V7) *47 A2 33UT*
TDS C Programmer's Guide *47 A2 07UT*

UFAS-EXTENDED User's Guide *47 A2 04UF*

IOF Terminal User's Reference Manual Part 1 (V6) *47 A2 31UJ*
IOF Terminal User's Reference Manual Part 2 (V6) *47 A2 32UJ*
IOF Terminal User's Reference Manual Part 3 (V6) *47 A2 33UJ*
IOF Terminal User's Reference Manual Part 4 (V6) *47 A2 34UJ*
IOF Terminal User's Reference Manual Part 1 (V7) *47 A2 38UJ*
IOF Terminal User's Reference Manual Part 2 (V7) *47 A2 39UJ*
IOF Terminal User's Reference Manual Part 3 (V7) *47 A2 40UJ*

GCL Programmer's Manual (V7) *47 A2 36UJ*

ACKNOWLEDGMENT

This acknowledgment has been reproduced from the CODASYL COBOL Journal of Development, 1984, as requested in that publication, prepared and published by the CODASYL Programming Language Committee.

"Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas from this report as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organization are requested to reproduce the following acknowledgment paragraphs in their entirety as part of the preface to any such publication. Any organization using a short passage from this document, such as in a book review, is requested to mention "COBOL" in acknowledgment of the source, but need not quote the acknowledgment.

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

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

> FLOW MATIC (trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No F 28 8013, copyrighted 1959 by IBM; FACT DSI 27A5260 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

Appendices

Illustrations

Figures

Tables

1. Concepts

1.1 INTRODUCTION

COBOL offers many features which allow the user to obtain a necessary function without programming the function in detail. In this chapter each of these features is discussed, considering the reason for its inclusion in the language and the concept of its use and organization.

1.2 FILES

A file is a collection of records which may be placed into or retrieved from a storage medium. The user not only chooses the file organization, but also chooses the file processing method and sequence. Although the file organization and processing method are restricted for sequential media, no such restrictions exist for mass storage media.

When describing the capabilities of COBOL programs to manipulate files, the following conventions are used. The term 'file-name' means the user-defined word used in the COBOL source program to reference a file. The terms 'file referenced by file-name' and 'file' mean the physical file regardless of the file-name used in the COBOL program. The term 'file connector' means the entity containing information concerning the file. All accesses to physical files occur through file connectors.

1.2.1 File Attributes

A file has several attributes which apply to the file at the time it is created and cannot be changed throughout the lifetime of the file. The primary attribute is the organization of the file, which describes its logical structure. There are three organizations: sequential, relative, and indexed. Other fixed attributes are prime record key, alternate record keys, code set, the minimum and maximum logical record size, the record type (fixed or variable), the collating sequence of the keys for indexed files, the blocking factor, the padding character, and the record delimiter.

1.2.1.1 Sequential Organization

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

A sequentially organized mass storage file has the same logical structure as a file on any sequential medium; however, a sequential mass storage file may be updated in place. When this technique is used, new records cannot be added to the file and each replaced record must be the same size as the original record.

1.2.1.2 Relative Organization

A file with relative organization is a mass storage file from which any record may be stored or retrieved by providing the value of its relative record number.

Conceptually, a file with relative organization comprises a serial string of areas, each capable of holding a logical record. Each of these areas is denominated by a relative record number. Each logical record in a relative file is identified by the relative record number of its storage area. For example, the tenth record is the one addressed by relative number 10 and is in the tenth record area, whether or not records have been written in any of the first through the ninth record areas.

In order to achieve more efficient access to records in a relative file, the number of character positions reserved on the medium to store a particular logical record may be different from the number of character positions in the description of that record in the program.

1.2.1.3 Indexed Organization

A file with indexed organization is a mass storage file from which any record may be accessed by giving the value of a specified key in that record. For each key data item defined for the records of a file, an index is maintained. Each such index represents the set of values from the corresponding key data item in each record. Each index, therefore, is a mechanism which can provide access to any record in the file.

Each indexed file has a primary index which represents the prime record key of each record in the file. Each record is inserted in the file, changed, or deleted from the file based solely upon the value of its prime record key. The prime record key of each record must be unique, and it must not be changed when updating a record. The prime record key is declared in the RECORD KEY clause of the file control entry for the file.

Alternate record keys provide alternative means of retrieval for the records of a file. Such keys are named in the ALTERNATE RECORD KEY clauses of the file control entry. The value of a particular alternate record key in each record need not be unique. When these values may not be unique, the DUPLICATES phrase is specified in the ALTERNATE RECORD KEY clause.

1.2.1.4 Logical Records

A logical record is the unit of data which is retrieved from or stored into a file. The number of records that may exist in a file is limited only by the capacity of the storage media. There are two types of records: fixed length and variable length. When the file is created, it is declared to contain either fixed length or variable length records. In any case, the content of a record area does not reflect any information added by the operating system, nor does the length of the record used by the COBOL programmer reflect these additions.

Fixed Length Records

Fixed length records must contain the same number of character positions for all the records in the file. All input-output operations on the file can process only this one record size. Fixed length records may be explicitly selected by specifying format 1 of the RECORD clause in the file description entry for the file regardless of the individual record descriptions |, or by specifying FLR in the file control entry for the file.|

Variable Length Records

Variable length records may contain differing numbers of character positions among the records on the file. To define variable length records explicitly, the VARYING phrase |or the DEPENDING phrase may be specified in the RECORD clause in the file description entry or the sort-merge file description entry for the file, or the VLR clause may be specified in the file control entry for the file.|The length of a record is affected by the data item referenced in the DEPENDING phrase of the RECORD clause or the DEPENDING phrase of an OCCURS clause or by the length of the record description entry for the file.

Implicit Record Types

When a file is not explicitly defined as fixed or variable length record (see the previous two paragraphs), it is implicitly fixed length except if one of the following conditions is true, in which case it is implicitly variable length record:

- the REPORT clause is specified for the file, or
- several record descriptions defining different record sizes are specified for the file, or
- a record description containing an OCCURS clause with the DEPENDING phrase is specified for the file;

1.2.2 File Processing

A file can be processed by performing operations upon individual records or upon the file as a unit. Unusual conditions that occur during processing are communicated back to the program.

1.2.2.1 Record Operations

The ACCESS MODE clause of the file control entry specifies the manner in which the object program operates upon records within a file. The access mode may be sequential, random, or dynamic.

For files that are organized as relative or indexed, any of the three access modes can be used to access the file regardless of the access mode used to create the file. A file with sequential organization may only be accessed in sequential mode.

The organization, format, and contents of an output report may be specified using the report writer feature. (See "Report Writer", this chapter.)

Sequential Access Mode

A file can be accessed sequentially irrespective of the file organization.

For sequential organization, the order of sequential access is the order in which the records were originally written.

For relative organization, the order of sequential access is ascending based on the value of the relative record numbers.

Only records which currently exist in the file are made available. The START statement may be used to establish a starting point for a series of subsequent sequential retrievals.

For indexed organization, the order of sequential access is ascending |or descending| based on the value of the key of reference according to the collating sequence of the file. Any of the keys associated with the file may be established as the key of reference during the processing of the file. The order of retrieval from a set of records which have duplicate key of reference values is the original order |or reverse of original order| of arrival of those records into the set. The START statement may be used to establish a starting point within an indexed file for a series of subsequent retrievals.

Random Access Mode

When a file is accessed in random mode, input-output statements are used to access the records in a programmer-specified order. The random access mode may only be used with relative or indexed file organizations.

For a file with relative organization, the programmer specifies the desired record by placing its relative record number in a relative key data item. With the indexed organization, the programmer specifies the desired record by placing the value of one of its record keys in a record key or an alternate record key data item.

Dynamic Access Mode

With dynamic access mode, the programmer may change at will from sequential accessing to random accessing, using appropriate forms of input-output statements. The dynamic access mode may only be used on files with relative or indexed organizations.

Open Mode

The open mode of the file is related to the actions to be performed upon records in the file. The open modes and purposes are: input, to retrieve records; output, to place records into a file; extend, to append records to an existing file; I-O to retrieve and update records. The open mode is specified in the OPEN statement.

When the open mode is input, a file may be accessed by a READ statement. The START statement may also be used for files organized as indexed or relative which are in sequential or dynamic access modes.

When the open mode is output, the records are placed into the file by issuing WRITE, GENERATE, or TERMINATE statements.

When the open mode is extend, new records are added to the logical end of a file by issuing WRITE, GENERATE, or TERMINATE statements.

Only mass storage files may be referenced in the open I-O mode. The additional capabilities of mass storage devices permits updating in place, thus READ and REWRITE statements may always be used. A mass storage file may be updated in the same manner as a file on a sequential medium, by transcribing the entire file into another file (perhaps in a separate area of mass storage) using READ and WRITE statements. However, it is sometimes more efficient to update a mass storage file in place. This mass storage file maintenance technique uses the REWRITE statement to return to their previous locations on the storage medium only those records which have changed. READ and REWRITE statements are the only operations allowed while updating in place sequentially organized files. However, for indexed or relative organized files, the following additional functions may be applied: the START statement may be used in sequential or dynamic access mode to alter the sequence of record retrieval; the DELETE statement may be used with any access mode to remove a record logically from a file; the WRITE statement may be used in random or dynamic access mode to insert a new record into the file.

Current Volume Pointer

The current volume pointer is a conceptual entity used in this document to facilitate exact specification of the current physical volume of a sequential file. The status of the current volume pointer is affected by the CLOSE, OPEN, READ, and WRITE statements.

File Position Indicator

The file position indicator is a conceptual entity used in this document to facilitate exact specification of the next record to be accessed within a given file during certain sequences of input-output operations. The setting of the file position indicator is affected only by the OPEN, READ, and START statements. The concept of a file position indicator has no meaning for a file opened in the output or extend mode.

Linage Concepts

The LINAGE clause may be used when specifying an output report. It facilitates definition of a logical page, and the positioning within that logical page of top and bottom margins and a footing area. Use of the LINAGE clause implicitly defines an associated special register, the LINAGE-COUNTER, which acts as a pointer to a line within the page body.

1.2.2.2 File Operations

Several COBOL statements operate upon files as entities or as collections of records. These are the CLOSE, MERGE, OPEN, and SORT statements.

Sorting

1. In many sort applications it is necessary to apply some special processing to the contents of a sort file. The special processing may consist of addition, deletion, creation, altering, editing, or other modification of the individual records in the file. It may be necessary to apply the special processing before or after the records are re-ordered by the sort, or special processing may be required in both places. The COBOL sort feature allows the user to express these procedures and to specify at which point, before or after the sort, they are to be executed. A COBOL program may contain any number of sorts, and each of them may have its own input and output procedures. The sort feature automatically causes execution of these procedures at the specified point.

2. Within an input procedure, the RELEASE statement is used to create the sort file. That is, at the completion of execution of the input procedure those records that have been processed by use of the RELEASE statement (rather than the WRITE statement) comprise the sort file, and this file is available only to the SORT statement. Execution of the SORT statement arranges the entire set of records in the sort file according to the keys specified in the SORT statement. The sorted records are made available from the sort file by use of the RETURN statement during execution of the output procedure.

3. The sort file has no label procedures which the programmer can control and the rules for blocking and for allocation of internal storage are peculiar to the SORT statement. The RELEASE and RETURN statements imply nothing with respect to buffer areas, blocks, or reels. A sort file, then may be considered as an internal file which is created (RELEASE statement) from the input file, processed (SORT statement), and then made available (RETURN statement) to the output file. The sort file itself is referred to and accessed only by the SORT statement. A sort-merge file description can be considered to be a particular type of file description. That is, a sort file, like any file, is a set of records.

Merging

1. In some applications it is necessary to apply some special processing to the contents of a merged file. The special processing may consist in addition, deletion, altering, editing, or other modification of the individual records in the file. The COBOL merge feature allows the user to express an output procedure to be executed as the merged output is created. The merged records are made available from the merged file by use of the RETURN statement in the output procedure.

2. The merge file has no label procedures which the programmer can control and the rules for blocking and for allocation of internal storage are peculiar to the MERGE statement. The RETURN statement implies nothing with respect to buffer areas, blocks, or reels.

3. A merge file, then, may be considered as an internal file which is created from input files by combining them (MERGE statement) as the file is made available (RETURN statement) to the output file. The merge file itself is referred to and accessed only by the MERGE statement. A sort-merge file description may be considered to be a particular type of file description. That is, a merge file, like any file, is a set of records.

1.2.2.3 Exception Handling

During the execution of any input or output operation, unusual conditions may arise which preclude normal completion of the operation. There are three methods by which these conditions are communicated to the object program: status keys, exception declaratives, and optional phrases associated with the imperative statement.

I-O Status

I-O status is a conceptual entity used in this document to facilitate exact specification of the status of the execution of an input-output operation. The setting of an I-O status is affected only by the CLOSE, DELETE, OPEN, READ, REWRITE, START, and WRITE statements. The I-O status value for a given file is made available to the program via the data-name specified in the FILE STATUS clause of the file entry for that file. The I-O status value is placed into this data item during the execution of the input-output statement and prior to the execution of any imperative statement associated with that input-output statement or prior to the execution of any exception declarative.

Exception Declaratives

A USE AFTER EXCEPTION procedure, when one is specified for the file, is executed whenever an input or output condition arises which results in an unsuccessful input-output operation. However, the exception declarative is not executed if the condition is invalid key and the INVALID KEY phrase is specified, or if the condition is at end and the AT END phrase is specified.

Optional Phrases

The INVALID KEY phrases may be associated with the DELETE, READ, REWRITE, START, or WRITE statements. Some of the conditions that give rise to an invalid key condition are when a requested key does not exist in the file (DELETE, READ, or START statements), when a key is already in a file and duplicates are not allowed (WRITE statement), and when a key does not exist in the file or when it was not the last key read (REWRITE statement). If the invalid key condition occurs during the execution of a statement for which the INVALID KEY phrase has been specified, the statement identified by that INVALID KEY phrase is executed.

The AT END phrase may be associated with a READ statement. The at end condition occurs in a sequentially accessed file when no next logical record exists in the file, when the number of significant digits in the relative record number is larger than the size of the relative key data item, when an optional file is not present, or when a READ statement is attempted and the at end condition already exists. If the at end condition occurs during the execution of a statement for which the AT END phrase has been specified, the statement identified by that AT END phrase is executed.

1.3 REPORT WRITER

The report writer is a special purpose feature which places its emphasis on the organization, format, and contents of an output report. Although a report can be produced using the standard COBOL language, the report writer language features provide a more concise facility for report structuring and report production. Much of the Procedure Division programming which would normally be supplied by the programmer is instead provided automatically by the report writer control system (RWCS). Thus the programmer is relieved of writing procedures for moving data, constructing print lines, counting lines on a page, numbering pages, producing heading and footing lines, recognizing the end of logical data subdivisions, updating sum counters, etc. All these operations are accomplished by the report writer control system as a consequence of source language statements that appear primarily in the Report Section of the Data Division of the source program.

1.3.1 Report Section

The Report Section of a COBOL Data Division contains one or more report description entries (RD entries), each of which forms the complete description of a report.

The report named in the report description entry is not assigned directly to an output file. Instead, it is associated with a file-name in the File Section and that file-name is associated with a file when an OPEN statement specifying the file-name is executed.

More than one report may be associated with the same file-name and the CODE clause is used to differentiate among the reports. For an external file connector referenced by a file-name, separately compiled programs may specify different reports for the same file-name. The file description entry of a file-name to which a report is assigned may not contain record description entries which describe data records. This file description entry must specify the name of a report description entry for each report associated with that file-name in this program.

The report description entry contains a set of clauses that names the report and supplies specific information about the format of the printed page and the organization of the subdivisions of the report.An identification code may be given in the report description entry so that each report may be identified separately in an intermediate output file.

Following each report description entry are one or more 01 level-number entries, each followed by a hierarchical structure similar to COBOL record descriptions. Each 01 level-number entry and its subordinate entries describes a report group. Each report group consists of zero, one, or more print lines that are regarded as a unit. A report group that is to be printed is printed entirely on one logical page; it is never split across pages.

1.3.2 Report Structure

When structuring a report, major consideration must be given to vertical and horizontal spacing requirements, manipulation of data, and the physical and logical subdivisions of a report.

1.3.2.1 Vertical Spacing

The report writer feature allows the user to describe report groups containing multiple lines. The vertical positioning of the lines on a page is specified by the LINE NUMBER clause that is associated with each line. The NEXT GROUP clause indicates how many lines to space after presenting the last line of the group. The first LINE NUMBER clause of the next group indicates additional spacing information to be used in positioning of that group.

1.3.2.2 Horizontal Spacing

The report writer allows the user to position the fields of data on a report line by means of the COLUMN NUMBER clause. The report writer control system supplies space fill between all defined fields.

1.3.2.3 Data Manipulation

When the report writer feature is used, data movement to a report group is directed by Report Section clauses rather than Procedure Divisions statements. The Report Section clauses which effect the manipulation of data are the SOURCE, SUM, and VALUE clauses.

The SOURCE clause specifies the sending data item of an implicit MOVE statement. The receiving printable item is defined by the description of the report group item in which the SOURCE clause appears.

The SUM clause automatically causes the establishment of a sum counter. The object of the SUM clause names the data item(s) which are added to the sum counter when a GENERATE statement is executed. The move of the sum counter contents to the receiving printable item, defined by the description of the report group item in which the SUM clause appears, is accomplished automatically when that report group is presented.

The VALUE clause defines a literal that appears in the printable item of a report group each time that report group is presented.

In summary, a data item in a report group is presented only if it has a COLUMN NUMBER clause specifying where it is to be presented.The value that is placed in a printable item is determined by the SOURCE, SUM, or VALUE clause stated in the report group description. Under no circumstances may a report group printable item receive a value directly via a Procedure Division statement.

1.3.2.4 Report Subdivisions

The physical and logical organization of a report interact to determine what is presented on a page.

Physical Subdivision of a Report

The PAGE clause specifies the length of the page, the size of the heading and footing areas, and the size of the area in which the detail lines will appear. The report writer control system uses the LINE number and NEXT group clauses to position these report groups, and when necessary, to advance to a new page with automatic production of PAGE HEADING and PAGE FOOTING report groups.

Logical Subdivision of a Report

Detail groups may be structured into a nested set of control groups.

Each control group may begin with a control heading and end with a control footing report group.

When nested control groups are defined, the recognition of a change in value of a control data item in a control hierarchy is called a control break and the heading and footing lines associated with the control data-name are called control heading and control footing report groups.

During the execution of a GENERATE statement, the report writer control system uses the control hierarchy to check automatically for control breaks. If a control break has occurred, all controls that are minor to it are considered to have changed, even though they may not in fact have changed. The occurrence of a control break causes the following sequence of events to take place:

1. All control footing report groups are presented up to and including the one at the level at which the control break occurred.

2. All control heading report groups are presented from the control break level down to the most minor control.

3. The detail report group named in the GENERATE statement is presented.

1.3.3 Procedure Division Report Writer Statements

The report writer statements that appear in the Procedure Division are: INITIATE, GENERATE, TERMINATE, SUPPRESS, and USE BEFORE REPORTING.

The INITIATE statement causes the report writer control system to perform automatically a number of initialization functions. A report must be initiated before any detail processing may take place.

The GENERATE statement which specifies a data-name causes the named DETAIL report group to be formatted and written to the output device. In addition, it triggers the report writer control system to perform the many implicit actions described above.

The GENERATE statement which specifies a report-name provides a means of summary reporting. A report produced by this type of statement has all detail print lines suppressed automatically and consists of only the summary totals accumulated during the processing of the DETAIL report group. The report writer control system processing for a GENERATE report-name statement is identical to that which occurs for a GENERATE data-name statement, except that the former results in the suppression of detail print lines.

The TERMINATE statement causes the report writer control system to perform all of the automatic functions associated with the termination of a report. The TERMINATE statement must be executed before the file containing the report is closed.

The SUPPRESS statement provides the object time facility to suppress the printing of an entire report group.

The BEFORE REPORTING phrase of the USE statement provides a mechanism whereby Procedure Division statements may be executed at specific instances within the automatic procedures performed by the report writer control system. The statements in the USE BEFORE REPORTING phrase may alter the contents of data items that are referenced by SOURCE clauses. Thus control is possible over the contents of data items referenced within report groups that are produced automatically.

1.4 TABLE HANDLING

Tables of data are common components of business data processing problems. Although the repeating items that make up a table could be otherwise described by a series of separate data description entries all having the same level-number and all subordinate to the same group item, there are two reasons why this approach is not satisfactory. First, from a documentation stand-point, the underlying homogeneity of the items would not be readily apparent; and second, the problem of making available an individual element of such a table would be severe when there is a decision as to which element is to be made available at object time.

Tables of data items are defined in COBOL by including an OCCURS clause in their data description entries. This clause specifies that the item is to be repeated as many times as stated. The item is considered to be a table element and its name and description apply to each repetition or occurrence. Since each occurrence of a table element does not have assigned to it a unique data-name, reference to a desired occurrence may be made only by specifying the data name of the table element together with the occurrence number of the desired table element. The occurrence number is known as a subscript.

The number of occurrences of a table element may be specified to be fixed or variable.

1.4.1 Table Definition

To define a one-dimensional table, the programmer uses an OCCURS clause as part of the data description of the table element, but the OCCURS clause must not appear in the description of group items which contain the table element. Example 1 shows a one-dimensional table defined by the item TABLE-ELEMENT.

Example 1:

```
01 TABLE-1.
   02 TABLE-ELEMENT OCCURS 20 TIMES.
      03 DOG...
      03 FOX...
```

In example 2, TABLE-ELEMENT defines a one-dimensional table, but DOG does not since there is an OCCURS clause in the description of the group item (TABLE-ELEMENT) which contains DOG.

Example 2:

```
02 TABLE-1.
   03 TABLE-ELEMENT OCCURS 20 TIMES.
      04 DOG OCCURS 5 TIMES.
         05 EASY...
         05 FOX...
```

In both examples, the complete set of occurrences of TABLE-ELEMENT has been assigned the name TABLE-1. However, it is not necessary to give a group name to the table unless it is desired to refer to the complete table as a group item.

None of the three one-dimensional tables which appear in the following two examples has a group name.

Example 3:

```
01 TABLE.
   02 BAKER...
   02 CHARLIE OCCURS 20 TIMES...
   02 DOG...
```

Example 4:

```
01 TABLE.
   02 BAKER OCCURS 20 TIMES...
   02 CHARLIE...
   02 DOG OCCURS 5 TIMES...
```

Defining a one-dimensional table within each occurrence of an element of another one-dimensional table gives rise to a two-dimensional table. To define a two-dimensional table, then, an OCCURS clause must appear in the data description of the element of the table, and in the description of only one group item which contains that table element. Thus, in example 5, DOG is an element of a two-dimensional table; it occurs 5 times within each element of the item BAKER which itself occurs 20 times. BAKER is an element of a one-dimensional table.

Example 5:

```
02 BAKER OCCURS 20 TIMES...
   03 CHARLIE...
   03 DOG OCCURS 5 TIMES...
```

In the general case, to define an n-dimensional table, the OCCURS clause should appear in the data description of the element of the table and in the description of (n-1) group items which contain the element.

1.4.2 Initial Values of Tables

In the Working-Storage or Constant Section, initial values of elements within tables are specified in one of the following ways:

1. The table may be described as a series of separate data description entries all subordinate to the same group item, each of which specifies the value of an element, or part of an element, of the table. In defining the record and its elements, any data description clause (USAGE, PICTURE, etc.) may be used to complete the definition, when required. The hierarchical structure of the table is then shown by use of the REDEFINES entry and its associated subordinate entries. The subordinate entries, following the REDEFINES entry, which are repeated due to OCCURS clauses, must not contain VALUE clauses.

2. All the dimensions of a table may be initialized by associating the VALUE clause with the description of the entry defining the entire table. The lower level entries will show the hierarchical structure of the table; lower level entries must not contain VALUE clauses.

1.4.3 References to Table Items

Whenever the user references a table element or a condition-name associated with a table element, the reference must indicate which occurrence of the element is intended, except in a USE FOR DEBUGGING statement and SEARCH statement. For access to a one-dimensional table the occurrence number of the desired element provides complete information. For tables of more than one dimension, an occurrence number must be supplied for each dimension of the table. In example 5, then, a reference to the fourth BAKER or the fourth CHARLIE would be complete, whereas a reference to the fourth DOG would not. To reference DOG, which is an element of a two-dimensional table, the user must reference, for example, the fourth DOG in the fifth BAKER.

1.4.4 Subscripting

Occurrence numbers are specified by appending one or more subscripts to the data-name.

The subscript can be represented either by an integer, a data-name which references an integer numeric elementary item or an index-name associated with the table |, or an arithmetic expression which produces an integer result. If such an arithmetic expression is used, it must be enclosed in parentheses|. A data-name or index-name may be followed by either the operator + or the operator - and an integer, which is used as an increment or decrement, respectively. It is permissible to mix integers, data-names, index-names |, and arithmetic expressions.|

The subscripts, enclosed in parentheses, are written immediately following any qualification for the name of the table element. |These parentheses are in addition to those, if any, that bound arithmetic expressions subscripts|. The number of subscripts in such a reference must equal the number of dimensions in the table whose element is being referenced. That is, there must be a subscript for each OCCURS clause in the hierarchy containing the data-name including the data-name itself.

When more than one subscript is required, they are written in the order of successively less inclusive dimensions of the data organization. If a multi-dimensional table is thought of as a series of nested tables and the most inclusive or outermost table in the nest is considered to be the major table with the innermost or least inclusive table being the minor table, the subscripts are written from left to right in the order major, intermediate, and minor.

A reference to an item must not be subscripted if the item is not a table element or an item or condition-name within a table element.

The lowest permissible occurrence number is 1. The highest permissible occurrence number in any particular case is the maximum number of occurrences of the item as specified in the OCCURS clause.

Using Integers, Data-Names |or Arithmetic Expressions|

When an integer, data-name |or an arithmetic expression is| used to represent a subscript, it may be used to reference items within different tables. These tables need not have elements of the same size. The same integer, data-name |or arithmetic expression| may appear as the only subscript with one item and as one of two or more subscripts with another item.

Using Index-Names

In order to facilitate such operations as table searching and manipulating specific items, a technique called indexing is available. To use this technique, the programmer assigns one or more index-names to an item whose data description entry contains an OCCURS clause. An index associated with an index-name acts as a subscript, and its value corresponds to an occurrence number for the item to which the index-name is associated.

The INDEXED BY phrase, by which the index-name is identified and associated with its table, is an optional part of the OCCURS clause. There is no separate entry to describe the index associated with index-name since its definition is completely hardware oriented. At object time the contents of the index correspond to an occurrence number for that specific dimension of the table with which the index is associated. The initial value of an index at object time is undefined, and the index must be initialized before use. The initial value of an index is assigned with the PERFORM statement with the VARYING phrase, the SEARCH statement with the ALL phrase, or the SET statement.

The use of an integer, a data-name, |or an arithmetic expression| as a subscript referencing a table element or an item within a table element does not cause the alteration of any index associated with that table.

An index-name can be used to reference only the table to which it is associated via the INDEXED BY phrase.

Data that is arranged in the form of a table is often searched. The SEARCH statement provides facilities for producing serial and non-serial (for example binary) searches. It is used to search a table for a table element that satisfies a specific condition and to adjust the value of the associated index to indicate that table element.

Relative indexing is an additional option for making references to a table element or to an item within a table element. When the name of a table element is followed by a subscript of the form (index-name + or - integer), the occurrence number required to complete the reference is the same as if index-name were set up or down by integer via the SET statement before the reference. The use of relative indexing does not cause the object program to alter the value of the index.

The value of an index can be made accessible to an object program by storing the value in an index data item. Index data items are described in the program by a data description entry containing a USAGE IS INDEX clause. The index value is moved to the index data item by the execution of a SET statement.

Example:

Assuming the following data definition:

```
02 XCOUNTER...

02 BAKER OCCURS 20 TIMES INDEXED BY BAKER-INDEX...
   03 CHARLIE...
   03 DOG OCCURS 5 TIMES...
      04 EASY
      88 MAX VALUE IS...
      04 FOX...
         05 GEORGE OCCURS 10 TIMES...
            06 HARRY...
            06 JIM...
```

References to BAKER and CHARLIE require only one subscript, references to DOG, EASY, MAX, and FOX require two, and references to GEORGE, HARRY and JIM require three.

To illustrate the requirement of order from major to minor, HARRY (18, 2, 7) means the HARRY in the seventh GEORGE, in the second DOG, in the eighteenth BAKER.

Mixing integers, data-names, and index-names is illustrated by HARRY (BAKER-INDEX, 4, XCOUNTER + 5).

1.5 SHARED MEMORY AREA

This feature is basically oriented toward saving memory space in the object program as it allows more than one file to share the same file area and input-output areas.

When the RECORD option of the SAME clause is used, only the record area is shared and the input-output areas for each file remain independent. In this case any number of the files sharing the same record area may be active at one time. This factor can give rise to an increase in the speed of the object program.

To illustrate this point, consider file maintenance. If the programmer assigns the same record area to both the old and new files, he not only saves memory in the object program, but because this technique eliminates a move of each record from the input to the output area, significant time savings result. An additional benefit of this technique is that the programmer need not define the record in detail as a part of both the old and new files. Rather, he defines the record completely in one case and simply includes the level 01 entry in the other. Because these record areas are in fact the same area, one set of names suffices for all processing requirements without requiring qualification.

1.6 PROGRAM AND RUN UNIT ORGANIZATION AND COMMUNICATION

Complete data processing problems are frequently solved by developing a set of separately compilable but logically coordinated programs which at some time prior to execution may be compiled and assembled into a complete problem solution. The organization of COBOL programs and run units supports this approach of dividing large problem solutions into small, more manageable, portions which may be programmed and validated independently.

1.6.1 Program and Run Unit Organization

there are two levels of computer programs in a COBOL environment. These are the source level and the object level.

At the source level, the most inclusive unit of a computer program is a source program. A source program may contain other source programs. A source program is a syntactically correct set of COBOL statements as specified in this document and consists of an Identification Division followed optionally by an Environment Division and/or a Data Division and/or a Procedure Division. A source program which itself is not contained within another source program |may optionally contain a Control Division. Such a program| can be converted by a compiler into an object program that either alone, or together with other object programs, is capable of being executed. In general, a source program which is contained within another program cannot itself be converted by a compiler into an object program, since the specifications in this document explicitly permit a contained source program to reference data in a containing source program.

The Procedure Division of a source program is organized into a sequence of procedures of two types. Declarative procedures, normally termed declaratives, are procedures which will be executed only when special conditions occur during the execution of a program. Non-declarative procedures are procedures which will be executed according to the normal flow of control within a program. Declaratives may contain non-declarative procedures but these will be executed only during the execution of the declaratives which contain them. Non-declarative procedures may contain other non-declarative procedures but must not contain a declarative. Neither declarative nor non-declarative procedures can contain programs. In other words, in COBOL the terms 'procedure' and 'program' are not synonyms.

At the object level the most inclusive unit of organization of computer programs is the run unit. A run unit is a complete problem solution consisting of an object program or of several inter-communicating object programs. A run unit is an independent entity that can be executed without communicating with, or being coordinated with, any other run unit except that it may process data files and messages or set and test switches that were written or will be read by other run units.

When a program is called, parameters upon which it is to operate may be passed to it by the program which calls it. As any separately compiled program may be the first program executed in a run unit, the first executed program of a run unit may receive parameters.

A run unit may also contain object code and data storage areas derived from the compilation of programs written in languages other than COBOL; in this case certain rules define the requirements for the relationship between the COBOL and the non-COBOL program. (See the *GCOS 7-V6 COBOL 85 User's Guide*.)

1.6.2 Accessing Data and Files

Some data items have associated with them a storage concept determining where data item values and other attributes of data items are represented with respect to the program of a run unit. Likewise, file connectors have associated with them a storage concept determining where information concerning the positioning and status of a file and other attributes of file processing are represented with respect to the program of a run unit.

1.6.2.1 Names

A data-name names a data item. A file-name names a file connector. These names are classified as either global or local.

A global name may be used to refer to the object with which it is associated either from within the program in which the global name is declared or from within any other program which is contained in the program which declares the global name.

A local name, however, may be used only to refer to the object with which it is associated from within the program in which the local name is declared. Some names are always global; other names are always local; and some other names are either local or global depending upon specifications in the program in which the names are declared.

A record-name is global if the GLOBAL clause is specified in the record description entry by which the record-name is declared, or, in the case of record description entries in the File |or Communication| Section, if the GLOBAL clause is specified in the file |or communication| description entry for the file-name |or cd-name| associated with the record description entry. A data-name is global if the GLOBAL clause is specified either in the data description entry by which the data-name is declared or in another entry to which that description is subordinate. A condition-name declared in a data description entry is global if that entry is subordinate to another entry in which the GLOBAL clause is specified.

A file-name is global if the GLOBAL clause is specified in the file description entry for that file-name.

|A cd-name is global if the GLOBAL clause is specified in the communication description entry for that cd-name.| A report-name is global if the GLOBAL clause is specified in the report description entry for that report-name.

However, specific rules sometimes prohibit specification of the GLOBAL clause for certain data description, file description, or record description entries.

If a name declared in a data description entry is not global, the name is local.

Global names are transitive across programs contained within other programs.

1.6.2.2 Objects

Accessible data items usually require that certain representations of data be stored. File connectors usually require that certain information concerning files be stored. The storage associated with a data item or a file connector may be external or internal to the program in which the object is declared.

Object Types

1. Working Storage Records

   Working storage records are allocations of sufficient storage to satisfy the record description entries in that section. Each record description entry in a program declares a different object. Renaming and redefining do not declare new objects; they provide alternate groupings or descriptions for objects which have already been declared.

2. File Connectors

   File connectors are storage areas which contain information about a file and are used as the linkage between a file-name and a physical file and between a file-name and its associated record area.

3. Record Areas for Files

   No particular record description entry in the File Section is considered to declare the storage area for the record.
   Rather, the storage area is the maximum required to satisfy associated record description entries. These entries may describe fixed or variable length records. In this presentation, record description entries are said to be associated in two cases. First, when record description entries are subordinate to the same file description entry, they are always associated. Second, when record description entries are subordinate to different file description entries and these file description entries are referenced in the same SAME RECORD AREA clause, they are associated. All associated record description entries are re-definitions of the same storage area.

4. Other Objects

   Examples of other objects declared in COBOL programs are: communication description entries, report description entries, and control information associated with the Communication, Linkage, and Report Sections.

Object Attributes

A data item or file connector is external if the storage associated with that object is associated with the run unit rather than with any particular program within the run unit. An external object may be referenced by any program which describes the object. References to an external object from different programs using separate descriptions of the object are always to the same object.

An object is internal if the storage associated with that object is associated only with the program that describes the object.

External and internal objects may have either global or local name.

1. Working-Storage Records

   A data record described in the Working-Storage Section is given the external attribute by the presence of the EXTERNAL clause in its data description entry. Any data item described by a data description entry subordinate to an entry describing an external record also attains the external attribute. If a record or data item does not have the external attribute, it is part of the internal data of the program in which it is described.

2. File Connectors

   A file connector is given the external attribute by the presence of the EXTERNAL clause in the associated file description entry. If the file connector does not have the external attribute, it is internal to the program in which the associated file-name is described.

3. Record Areas for Files

   The data records described subordinate to a file description entry which does not contain the EXTERNAL clause or a sort-merge file description entry, as well as any data items described subordinate to the data description entries for such records, are always internal to the program describing the file-name. If the EXTERNAL clause is included in the file description entry, the data records and the data items attain the external attribute.

4. Other Objects

   Data records, subordinate data items, and various associated control information described in the Linkage, Communication, and Report Sections of a program are always considered to be internal to the program describing that data. Special considerations apply to data described in the Linkage Section whereby an association is made between the data records described and other data items accessible to other programs. (See "Passing Parameters to Programs", this chapter.)

1.6.2.3 Name Resolution

Certain conventions apply when programs contained within other programs assign the same names to data items, conditions, and file connectors. Consider the situation when program A contains program B which itself contains program C; further, programs A and B, but not program C, contain Data Division entries for a condition-name, data-name, or a file-name named DUPLICATE-NAME.

1. If either DUPLICATE-NAME references an internal object, two different, though identically named objects exist. If both DUPLICATE-NAMEs reference an external object, only one object exists.

2. Program A's reference to DUPLICATE-NAME is always to the object which it declares. Program B's reference to DUPLICATE-NAME is always to the object which it declares.

3. If DUPLICATE-NAME is a local name in both programs A and B, program C cannot refer to that name.

4. If DUPLICATE-NAME in program B is a global name, program C may access the object referenced by the name in program B, regardless of whether or not DUPLICATE-NAME is a global name in program A.

5. If DUPLICATE-NAME in program A is a global name but in program B it is a local name, program C's reference to DUPLICATE-NAME is to the object referenced by the name declared in program A.

1.6.3 Program Classes

All programs which form part of a run unit may possess none, one, or more of the following attributes: common and initial.

1.6.3.1 Common Programs

A common program is one which, despite being directly contained within another program, may be called by any program directly or indirectly contained in that other program. The common attribute is attained by specifying the COMMON phrase in a program's Identification Division. The COMMON phrase facilitates the writing of subprograms which are to be used by all the programs contained within a program.

1.6.3.2 Initial Programs

An initial program is one whose program state is initialized when the program is called. Thus, whenever an initial program is called, its program state is the same as when the program was first called in that run unit. During the process of initializing an initial program, that program's internal data is initialized; thus an item of the program's internal data whose description contains a VALUE clause is initialized to that defined value, but an item whose description does not contain a VALUE clause is initialized to an undefined value. Files with internal file connectors associated with the program are not in the open mode. The control mechanisms for all PERFORM statements contained in the program are set to their initial states. The initial attribute is attained by specifying the INITIAL phrase in the program's Identification Division.

1.6.4 Inter-Program Communication

When the complete solution to a data processing problem is subdivided into more than one program, the constituent programs must be able to communicate with each other. This communication may take four forms: the transfer of control, the passing of parameters, the reference to common data, and the reference to common files. These four inter-program communication forms are provided both when the communicating programs are separately compiled and when one of the communicating programs is contained within the other program. The precise mechanisms provided in the last two cases differ from those in the first two cases; for example, a program contained within another program may reference a data-name or file-name possessing a global name in the containing program. (See "Names", this chapter.)

1.6.4.1 Transfer of Control

The CALL statement provides the means whereby control may be transferred from one program to another program within a run unit. A called program may itself contain CALL statements.

When control is transferred to a called program, execution proceeds from statement to statement beginning with the first non-declarative statement. If control reaches a STOP RUN statement, this signals the logical end of the run unit. If control reaches an EXIT PROGRAM statement, this signals the logical end of the called program only, and control then reverts to the next executable statement following the CALL statement in the calling program. Thus the EXIT PROGRAM statement terminates only the execution of the program in which it occurs, while the STOP RUN statement terminates the execution of a run unit.

The CALL statement may be used to call a program which is not written in COBOL. A COBOL program may also be called from a program which is not written in COBOL. In both cases, only those parts of the parameter passing mechanism which apply to the COBOL program are specified in this document. (For more details, refer to the *COBOL 85 User's Guide*, Calling and Called Programs).

Names of Programs

In order to call a program, a CALL statement identifies the program's name. The names assigned to programs which directly or indirectly are contained within another program must be unique.

The names assigned to each of the separately compiled programs which constitute a run unit must also be unique.

Scope of the CALL Statement

In the following, the calling program may or may not possess any of the program attributes, it may either be separately compiled or not, and it may either be contained within programs or contain programs:

1. Any calling program may call any separately compiled program in the run unit.

2. A calling program may call any program which is directly contained within the calling program.

3. Any calling program may call any program possessing the common attribute which is directly contained within a program which itself directly or indirectly contains the calling program, unless the calling program is itself contained within the program possessing the common attribute.

4. A calling program may call a program which neither possesses the common attribute nor is separately compiled if, and only if, that program is directly contained within the calling program.

Scope of Names of Programs

Certain conventions apply when, within a separately compiled program, a name identical to that specified for another separately compiled program in the run unit is specified for a contained program.

Consider the situation when program A contains program B and program DUPLICATE-NAME, program B contains program BB, and program DUPLICATE-NAME contains program DD.

The name DUPLICATE-NAME has also been specified for a separately compiled program.

1. If program A, but not any of the programs it contains, calls program DUPLICATE-NAME, the program activated is the one contained within program A.

2. If either program B or program BB calls program DUPLICATE-NAME then:

   a. If the program DUPLICATE-NAME contained within program A possesses the common attribute, it is called.

   b. If the program DUPLICATE-NAME contained within program A does not possess the common attribute, the separately compiled program is called.

3. If either program DD or program DUPLICATE-NAME contained within program A calls program DUPLICATE-NAME, the program called is the separately compiled program.

4. If any other separately compiled program in the run unit or any other program contained within such a program calls the program DUPLICATE-NAME, the program called is the separately compiled program named DUPLICATE-NAME.

1.6.4.2 Passing Parameters to Programs

A program calls another program in order to have the called program perform, on behalf of the calling program, some defined part of the solution of a data processing problem. In many cases it is necessary for the calling program to define to the called program the precise part of the problem solution to be executed by making certain data values, which the called program requires, available to the called program. One method for ensuring the availability of these data values is by passing parameters to a program, as is described in this paragraph. Another method is to share the data. (See below.) The data values passed as parameters also may identify some data to be shared; hence the two methods are not mutually independent.

Identifying Parameters

Data passed as parameter by a program calling another program must be accessible to the calling program and the data item receiving the data must be declared in the Data Division of the called program. In the called program the parameters required are identified by listing references to the names assigned, in that program's data description entries, to the parameters in that program's Procedure Division header. In the calling program the values of the parameters to be passed by the calling program are identified by listing references in the CALL statements used to call the called program. These lists establish, on a positional basis at object time, the correspondence between the values as they are known to each program; that is, the first parameter on one list corresponds to the first parameter on the other, the second to the second, etc. Thus a program, which may be called by another program, may include:

```
PROGRAM-ID. EXAMPLE.

PROCEDURE DIVISION USING NUM, PCODE, COST.
```

and may be called by executing:

```
CALL "EXAMPLE" USING NBR, PTYPE, PRICE.
```

thereby establishing the following correspondence:

Called Program (EXAMPLE)	Calling Program
NUM	NBR
PCODE	PTYPE
COST	PRICE

Only the positions of the data-names are significant, not the names themselves.

Values of Parameters

The calling program controls the methods by which a called program evaluates the values of the parameters passed to it and by which the called program returns results as modified parameter values.

The individual parameters referenced in the CALL statement's USING phrase may be passed either by reference or by content. A called program is allowed to access and modify the value of the data item referenced in the calling program's CALL statement as a parameter passed by reference. This permission to access and modify a data item in the calling program is denied to the called program if the data item is specified in the CALL statement as a parameter passed by content. The value of the parameter is evaluated when the CALL statement is executed and is presented to the called program. This value may be changed by the called program during the course of its execution, but the value of the corresponding data item in the calling program is not modified. Thus a parameter passed by reference may be used by a called program to return a result to the calling program whereas a parameter passed by content cannot be so used.

The parameters referenced in a called program's Procedure Division header must be described in the Linkage Section of that program's Data Division.

1.6.4.3 Sharing Data

Two programs in a run unit may reference common data in the following circumstances:

1. The data content of an external data record may be referenced from any program provided that program has described that data record. (See above, "Objects").

2. If a program is contained within another program, both programs may refer to data possessing the global attribute either in the containing program or in any program which directly or indirectly contains the containing program. (See above, "Names".)

3. The mechanism whereby a parameter value is passed by reference from a calling program to a called program establishes a common data item; the called program, which may use a different identifier, may refer to a data item in the calling program.

1.6.4.4 Sharing Files

Two programs in a run unit may reference common file connectors in the following circumstances:

1. An external file connector may be referenced from any program which describes that file connector. (See above, "Objects".)

2. If a program is contained within another program, both programs may refer to a common file connector by referring to an associated global file-name either in the containing program or in any program which directly or indirectly contains the containing program. (See above, "Names".)

1.6.5 Intra-Program Communication

The procedures which constitute the Procedure Division of a program communicate with one another by transferring control or by referring to common data.

1.6.5.1 Transfer of Control

There are four methods of transferring control within a program:

1. A GO TO statement.
2. A PERFORM statement.
3. An input procedure associated with a SORT statement, or an output procedure associated with a SORT or a MERGE statement.
4. A declarative procedure which is activated whenever certain conditions, including errors and exceptions, occur.

An input-output procedure can be considered as an implicit PERFORM statement which is executed in conjunction with a SORT or MERGE statement; and, for this reason, the restrictions on the PERFORM statement apply equally to input-output procedures.

Stricter restrictions than those for the PERFORM statement apply to declarative procedures.

1.6.5.2 Shared Data

All the data declared in a program's Data Division may be referenced by statements in the procedures, input-output procedures, and declaratives which constitute that program. Under certain conditions a program may reference data items whose declarations are not included in its Data Division. (See above, "Accessing Data and Files".)

1.6.6 Segmentation

The segmentation facility permits the user to subdivide physically the Procedure Division of a COBOL object program. All source paragraphs which contain the same segment-number in their section headers will be considered at object time to be one segment. Since segment-numbers can range from 00 through 99, it is possible to subdivide any object program into a maximum of 100 segments.

Program segments may be of three types: fixed permanent, fixed overlayable, and independent as determined by the programmer's assignment of segment-numbers.

Fixed segments are always in computer storage during the execution of the entire program; i.e., they cannot be overlayed except when the system is executing another program, in which case fixed segments may be 'rolled out' temporarily.

Fixed overlayable segments may be overlayed during program execution, but any such overlaying is transparent to the user, i.e., they are logically identical to fixed segments, but physically different from them.

Independent segments may be overlayed, but such overlaying will result in the initialization of those segments. Therefore, independent segments are logically different from fixed permanent/fixed overlayable segments, and physically different from fixed segments.

1.7 COMMUNICATION FACILITY

The communication facility provides the ability to access, process, and create messages or portions thereof. It provides the ability to communicate through a message control system with local and remote communication devices.

1.7.1 MCS (Message Control System)

The implementation of the communication facility requires that a message control system (MCS) be present in the COBOL object program's environment.

The message control system (MCS) is the logical interface to the operating system under which the COBOL object program operates. The primary functions of the message control system are the following:

1. To act as an interface between the COBOL object program and the network of communication devices, in much the same manner as an operating system acts as an interface between the COBOL object program and such devices as card readers, printers, magnetic tape, and mass storage devices.

2. To perform line discipline, including such tasks as dial-up, polling, and synchronization.

3. To perform device-dependent tasks, such as character translation and insertion of control characters, so that the COBOL user can create device-independent programs.

The first function, that of interfacing the COBOL object program with the communication devices, is the most obvious to the COBOL user. In fact, the COBOL user may be totally unaware that the other two functions exist. Messages from communication devices are placed in input queues by the message control system while awaiting disposition by the COBOL object program. Output messages from the COBOL object program are placed in output queues by the message control system while awaiting transmission to communication devices. The structures, formats, and symbolic names of the queues are defined by the user to the message control system at some time prior to the execution of the COBOL object program. Symbolic names for message sources and destinations are also defined at that time. The COBOL user must specify in his COBOL program symbolic names which are known to the message control system.

During execution of a COBOL object program, the message control system performs all necessary actions to update the various queues as required.

1.7.2 The COBOL Object Program

The COBOL object program interfaces with the message control system when it is necessary to send data, receive data, or to interrogate the status of the various queues which are created and maintained by the message control system. In addition, the COBOL object program may direct the message control system to establish or break the logical connection between the communication device and a specified portion of the message control system queue structure. The method of handling the physical connection is a function of the message control system.

1.7.3 Relationship to MCS and Communication Devices

The interfaces which exist in a COBOL communication environment are established by the use of a communication description entry (CD entry) in the Communication Section of the Data Division. There are two such interfaces:

1. The interface between the COBOL object program and the message control system, and;

2. The interface between the message control system and the communication devices.

The COBOL source program uses three statements to control the interface with the message control system:

1. The RECEIVE statement, which causes data in a queue to be passed to the COBOL object program,

2. The SEND statement, which causes data associated with the COBOL object program to be passed to one or more queues, and;

3. The ACCEPT MESSAGE COUNT statement, which causes the message control system to indicate to the COBOL object program the number of complete messages in the specified queue structure.

The COBOL source program uses two statements to control the interface between the message control system and the communication devices:

1. The ENABLE statement, which establishes logical connection between the message control system and one or more given communication devices, and;

2. The DISABLE statement, which breaks a logical connection between the message control system and one or more given communication devices.

These relationships are shown in Figure 1-1, COBOL Communication Environment, and explained below (Enabling and Disabling Queues).

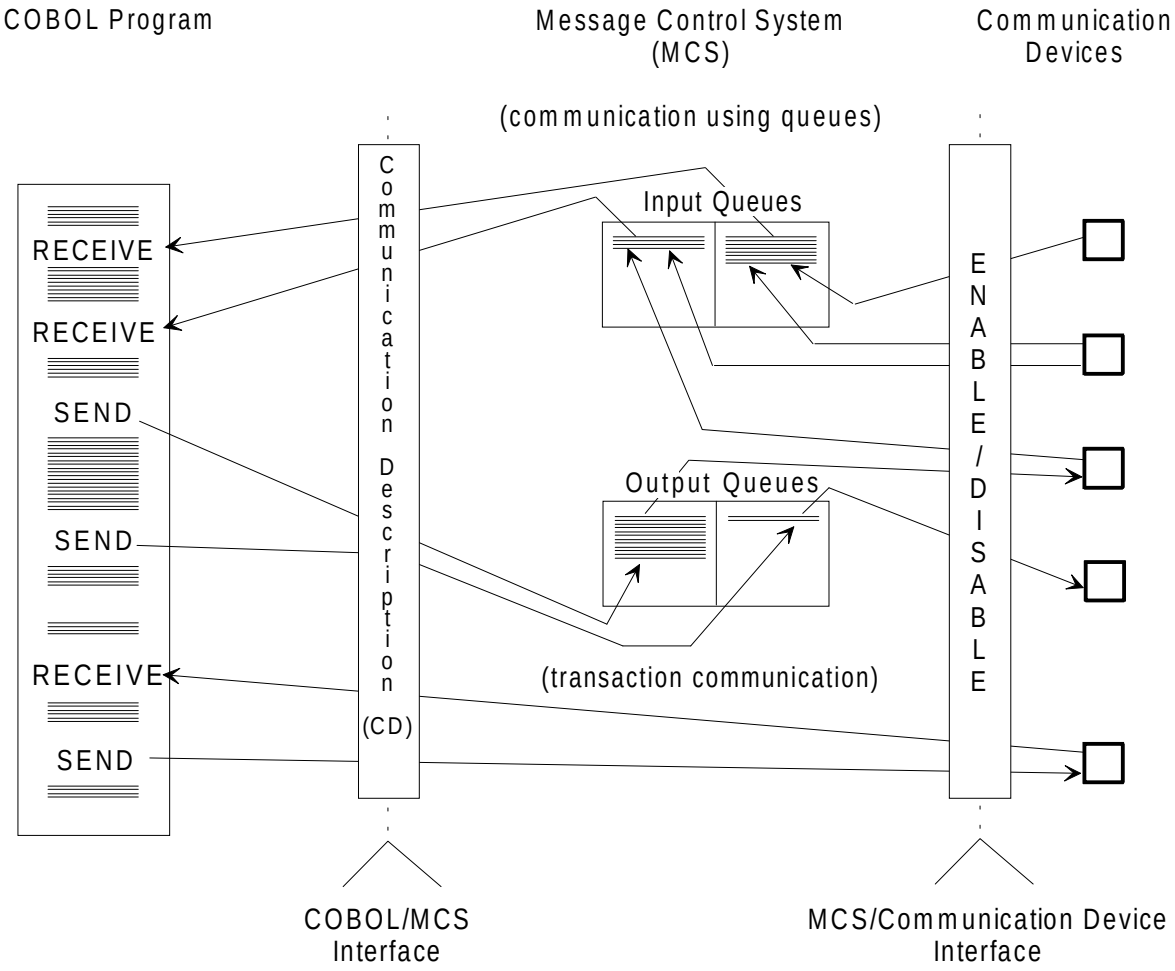


Figure 1-1. COBOL Communication Environment

Invoking the COBOL Object Program

There are two methods of invoking a COBOL communication object program: scheduled initiation and message control system (MCS) invocation. Regardless of the method of invocation, the only operating difference between the two methods is that MCS invocation causes certain areas in the referenced communication description entry (CD entry) to be filled.

Scheduled Initiation of the COBOL Program

A COBOL object program using the communication facility may be scheduled for execution through the normal means available in the program's operating environment, such as job control language. In that case, the COBOL program can use three methods to determine what messages, if any, are available in the input queues:

1. The ACCEPT MESSAGE COUNT statement,

2. The RECEIVE statement with a NO DATA phrase, and

3. The RECEIVE statement without a NO DATA phrase (in which case a program wait is implied if no data is available).

Invocation of the COBOL Object Program by MCS

It is sometimes desirable to schedule a COBOL object communication program only when there is work available for it to do. Such scheduling occurs if the message control system (MCS) determines what COBOL object program is required to process the available message and subsequently causes that program to be scheduled for execution. Each object program scheduled by the MCS establishes a run unit. Prior to the execution of the COBOL object program, the message control system places the symbolic queue and sub-queue names in the associated data items of the communication description entry that specifies the FOR INITIAL INPUT clause, or the message control system places the symbolic terminal name in the associated data item of the communication description entry that specifies the FOR INITIAL I-O clause.

A subsequent RECEIVE statement directed to that communication description entry will result in the available message being passed to the COBOL object program.

Determining the Method of Scheduling

A COBOL source program can be written so that its object program can operate with either of the above two modes of scheduling. In order to determine which method was used to load the COBOL object program, the following is one technique that may be used:

1. One communication description entry (CD entry) must contain a FOR INITIAL I-O clause.

2. When the program contains a CD with the FOR INITIAL INPUT clause, the Procedure Division may contain statements to test the initial value of the symbolic queue name in that communication description entry. If it is space filled, job control statements were used to schedule the COBOL object programs. If not space filled, the message control system has invoked the COBOL object program and initialized the data item with the symbolic name of the queue containing the message to be processed.

3. When the program contains a CD entry with the FOR INITIAL I-O clause, the Procedure Division may contain statements to test the initial value of the symbolic terminal name in that CD.
   If it is space filled, job control statements were used to schedule the COBOL object program. If not space filled, the MCS has invoked the COBOL object program and initialized the data item with the symbolic name of the communication terminal that is source of the message to be processed.

1.7.4 The Concept of Messages and Message Segments

A message consists of some arbitrary amount of information, usually character data, whose beginning and end are defined or implied. As such, messages comprise the fundamental but not necessarily the most elementary unit of data to be processed in a COBOL communication environment.

Messages may be logically subdivided into smaller units of data called message segments which are delimited within a message by means of end of segment indicators (ESI). A message consisting of one or more segments is delimited from the next message by means of an end of message indicator (EMI). In a similar manner, a group of several messages may be logically separated from succeeding messages by means of an end of group indicator (EGI).

When a message or message segment is received by the COBOL program, a communication description interface area is updated by the message control system to indicate which, if any, delimiter was associated with the text transferred during the execution of that RECEIVE statement.

On output the delimiter, if any, to be associated with the text released to the message control system during execution of a SEND statement is specified or referenced in the SEND statement. Thus the presence of these logical indicators is recognized and specified both by the message control system and by the COBOL object program; however, no indicators are included in the message text processed by COBOL programs.

A precedence relationship exists between the indicators EGI, EMI, and ESI. EGI is the most inclusive indicator and ESI is the least inclusive indicator. The existence of an indicator associated with message text implies the association of all less inclusive indicators with that text. For example, the existence of the EGI implies the existence of EMI and ESI.

1.7.5 The Concept of Queues

The following discussion applies only when the COBOL communication environment is established using a communication description entry without the FOR I-O clause.

Queues consist of one or more messages from or to one or more communication devices, and as such, form the data buffers between the COBOL object program and the message control system. Input queues are logically separate from output queues.

The message control system logically places in queues or removes from queues only complete messages. Portions of messages are not logically placed in queues until the entire message is available to the message control system. That is, the message control system will not pass a message segment to a COBOL object program unless all segments of that message are in the input queue; even though the COBOL source program uses the SEGMENT phrase of the RECEIVE statement. For output messages, the message control system will not transmit any segment of a message until all its segments are in the output queue. Interrogation of the queue depth, or number of messages that exist in a given queue, reflects only the number of complete messages that exist in the queue.

The process by which messages are put into a queue is called enqueueing. The process by which messages are removed from a queue is called dequeueing.

Independent Enqueueing and Dequeueing

It is possible that a message may be received by the message control system from a communication device prior to the execution of the COBOL object program. As a result, the message control system enqueues the message in the proper input queue (provided that input queue is enabled) until the COBOL object program requests dequeueing with the RECEIVE statement. It is also possible that a COBOL object program will cause the enqueueing of messages in an output queue which are not transmitted to a communication device until after the COBOL object program has terminated. Two common reasons for this occurrence are:

1. When the output queue is disabled.

2. When the COBOL object program creates output messages at a speed faster than the destination can receive them.

Enabling and Disabling Queues

Usually, the message control system will enable and disable queues based on time of day, message activity, or other factors unrelated to the COBOL program. However, the COBOL program has the ability to enable and disable queues itself through use of the ENABLE and DISABLE statements.

Enqueueing and Dequeueing Methods

In systems that allow the user to specify certain MCS functions, it may be necessary that the user specify to the message control system, prior to execution of programs which reference these facilities, the selection algorithm and other designated MCS functions to be used by the message control system in placing messages in the various queues. A typical selection algorithm for example would specify that all messages from a given source be placed in a given input queue, or that all messages to be sent to a given destination be placed in a given output queue.

Dequeueing is often done on a first in, first out basis. Thus messages dequeued from either an input or output queue are those messages which have been in the queue for the longest period of time. However, the message control system can, upon prior specification by the user, dequeue on some other basis, e.g., priority queueing can be employed.

Queue Hierarchy

In order to control more explicitly the messages being enqueued and dequeued, it is possible to define in the message control system a hierarchy of input queues, i.e., queues comprising queues. In COBOL, four levels of queues are available to the user. In order of decreasing significance, the queue levels are named queue, sub-queue-1, sub-queue-2, and sub-queue-3. The full queue structure is depicted in the figure "Hierarchy of Queues" below, where queues and sub-queues have been named with the letters A through O. Messages have been named with a letter according to their source (X, Y, or Z) and with a sequential number.

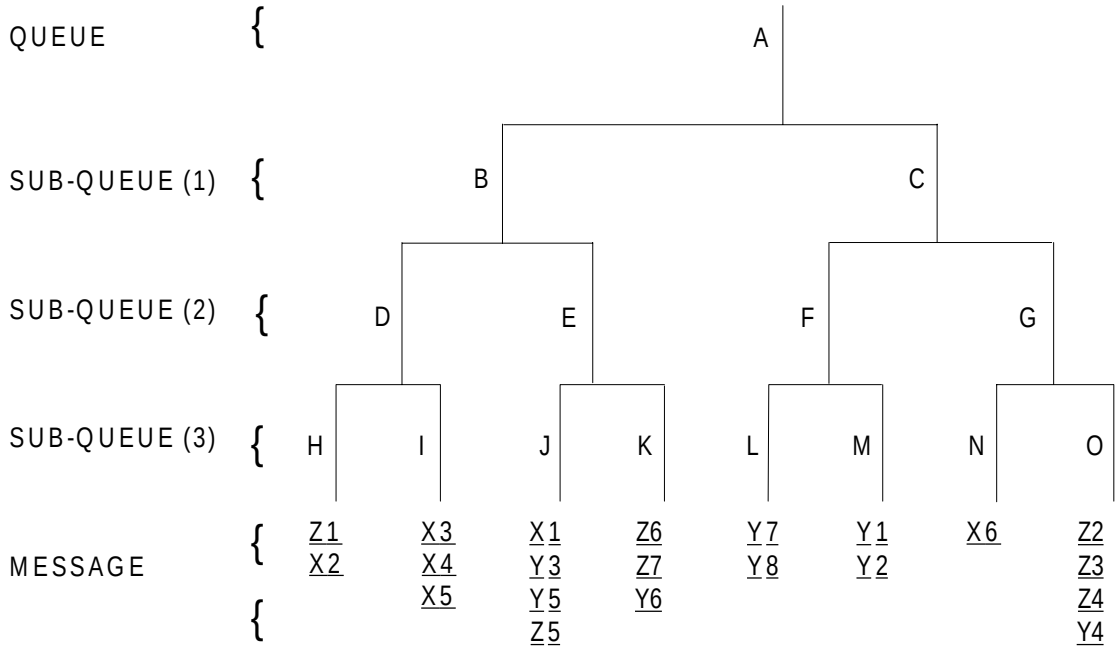


Figure 1-2. Hierarchy of Queues

Let us assume that the message control system is operating under the following queueing algorithm:

1. Messages are placed in queues according to the contents of some specified data field in each message.

2. With the RECEIVE statement, if the user does not specify a given sub-queue level, the message control system will choose the sub-queue from that level in the alphabetic order, e.g., if sub-queue-1 is not specified by the user, the message control system will dequeue from sub-queue-1 B.

The following examples illustrate the effect of the above algorithm (see the figure "Hierarchy of Queues" above):

1. The program executes a RECEIVE statement, specifying via the communication description entry:

```
Queue A
Message control system returns: Message Z1
```

2. The program executes a RECEIVE statement, specifying via the communication description entry:

```
Queue A
Sub-queue-1 C
Message control system returns: Message Y7
```

3. The program executes a RECEIVE statement, specifying via the communication description entry:

```
Queue A
Sub-queue-1 B
Sub-queue-2 E
Message control system returns: Message X1
```

4. The program executes a RECEIVE statement, specifying via the communication description entry:

```
Queue A
Sub-queue-1 C
Sub-queue-2 G
Sub-queue-3 N
Message control system returns: Message X6
```

If the COBOL programmer wishes to access the next message in a queue, regardless of which sub-queue that message may be in, he specifies the queue name only. The message control system, when supplying the message, will return to the COBOL object program any applicable sub-queue names via the data items in the associated communication description entry. If, however, he desires the next message in a given sub-queue, he must specify both the queue name and any applicable sub-queue names.

For output, the COBOL user specifies only the destination(s) of the message, and the message control system places the message in the proper queue structure.

There is no one-to-one relationship between a communication device and a source/destination. A source or destination may consist of one or more physical device(s). The device or devices which comprise a source/destination are defined to the message control system.

1.7.6 The Concept of Transaction Communication

In contrast with the previously described queueing mechanism, some applications require a direct dialogue between a communication device and the object program. In this case, it is unnecessary to queue messages for processing since they are to be processed immediately. It is possible in COBOL to specify this kind of processing by using the CD that specifies the FOR I-O clause. A CD that specifies the FOR I-O clause can communicate with only one terminal; however, a run unit may contain more than one CD that specifies the FOR I-O clause and these CDs can communicate with the same or a different terminal. When the INITIAL phrase is used in a CD that specifies the FOR I-O clause, the program may be scheduled by the MCS.

1.8 INTRINSIC FUNCTION FACILITY

Data processing problems frequently require the use of values which are not directly accessible in the data storage associated with the object program. These data values must be derived through performing some operations on other data. An intrinsic function represents a temporary data item whose value is derived automatically at the time of reference during the execution of object program.

The value returned by a function is considered to be a data value. A mechanism is provided at object time to assign a data value to a function when it is referenced. In order to determine the value of a function, the evaluation mechanism may require access to data values provided by the referencing program. These data values are provided by specifying parameters, known as arguments, when referencing the function. Specific functions may place constraints on these arguments such as range, etc. If, at the time a function is referenced, the arguments specified for that reference do not have values that comply with the specified constraints, the returned value for the function is undefined.

2. Notation Used in Formats and Rules

2.1 DEFINITION OF A GENERAL FORMAT

In this manual, the general format is the specific arrangement of the elements of a clause or a statement, followed by 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 can appear in sequences other than that shown. Applications, requirements or restrictions are shown as rules.

Throughout this document, specifications that do not pertain to the American National Standard COBOL 1985 are enclosed in boxes.

The following types of rules appear adjacent to each format in this manual:

1. Syntax Rules: 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 can represent.

2. General Rules: Those rules that define or clarify the semantics of the statement and the effect of the statement on execution or compilation.

2.2 FORMATS ELEMENTS

Elements that make up a clause or a statement consists of uppercase words, lower-case words, level-numbers, brackets, braces, connectives and special characters.

2.2.1 Upper-case and Lower-case Words

The underlined uppercase words in the format are called keywords and are required when the functions of which they are a part are used.

Uppercase words that are not underlined are optional to the user and may or may not be present in the source program. Uppercase words, whether underlined or not, must be spelled correctly.

Lower-case words represent information to be supplied by the user. In a general format, the lower-case words are generic terms used to represent COBOL words, literals, PICTURE character-strings, comment-entries, or a complete syntactical entry that must be furnished by the user. Where generic terms are repeated in a general format, a number or letter appendage to the term serves to identify that term for explanation or discussion (for example, identifier-1, identifier-2).

The rules governing the use of characters in COBOL words, character-strings, literals, and other source program entries are given in Chapter 3.

2.2.2 Level-Numbers

When specific level-numbers appear in data description entry formats, those specific level-numbers are required when such entries are used in a COBOL program. In this manual the form 01, 02, 03, 04, 05, 06, 07, 08, 09 is used to indicate level-numbers 1 through 9. The leading zero is optional.

2.2.3 Brackets and Braces

When a portion of a general format is enclosed in brackets, [ ], it is optional, and may be included or omitted according to the user's objectives. When a portion of a general format is enclosed in braces, { }, one of the options contained within the braces must be selected. In both cases, a choice is indicated by vertically stacking the possibilities. When brackets or braces enclose a portion of a format, but only one possibility is shown, the function of the brackets or braces is to delimit that portion of the format to which a following ellipsis applies (See "Ellipsis", below).

2.2.4 Ellipsis

In the text of this manual, the ellipsis (...) may show the omission of a portion of a source program; this meaning becomes apparent in context. In general formats, the ellipsis represents the position at which repetition can occur at the user's option.

The option of the format that can be repeated is determined as follows:

1. given ... in a clause or statement format,

2. scan right to left and determine the] or } immediately to the left of the ... ,

3. Continue scanning right to left and determine the logically matching [or {,

4. then, the ... applies to the words between the determined pair of delimiters.

2.3 FORMAT PUNCTUATION

The separators comma and semi-colon may be used to improve the readability of the program. Their use is optional and they are not shown in formats. In the source program, separators comma, semi-colon, |'Horizontal Tabulation'| and space are interchangeable.

If desired, a semi-colon or comma may be used between statements in the Procedure Division.

Paragraphs within the Identification and Procedure Divisions and entries within the Environment and Data Division must be terminated by the separator period.

2.4 USE OF SPECIAL CHARACTER WORDS IN FORMATS

The special character words "+", "-", "<", ">", "=", ">=", "<=", when appearing in formats, although not underlined, are required when such portions of the formats are used.

3. COBOL Language Concepts

3.1 COBOL CHARACTER SET

The most basic and indivisible unit of the language is the character. The set of characters used to form COBOL character-strings and separators consists of the digits 0 through 9, the 26 uppercase and 26 lower-case letters of the English alphabet, the space (blank), and special characters; all are listed in Part 1 of the table "COBOL Characters" below.

The characters used for punctuation are shown in Part 2 of the table "COBOL Characters" below. Editing characters, which may be a single character or a fixed 2-character combination, are shown in Part 3 of the table. Characters used in relation conditions are listed in Part 4 of the table.

The following characters are used to form COBOL words:

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
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
0 1 2 3 4 5 6 7 8 9
- (hyphen)
_ (underscore)

Non-numeric literals, comment-entries and comment lines can include any character of the computer's entire character set.

|Unless the user employs the NCASEQ option in the COBOL JCL statement, all lower-case letters used while writing a source program, except those in non-numeric literals, and except the symbols 'f', 'g', 'h', 'i', 'j', 'k', 'm', 'n', 'o', 'q', 't', 'u', 'w', 'y', and possibly any lower-case currency sign of PICTURE character strings, are treated as uppercase by the compiler.|

|If the user employs the NCASEQ option in the COBOL JCL statement, only lower-case letters used while writing reserved words and the symbols 'a', 'b', 'c', 'd', 'e', 'l', 'p', 'r', 's', 'v', 'x' and 'z' of PICTURE characters-strings are treated as uppercase by the compiler. Lower-case letters will appear in lower-case form in the output only if the printer configuration has lower-case capability.|

COBOL Characters

Table 3-1. The Complete COBOL Character Set

Characters	Meaning
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	upper-case 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	lower-case letters
0 1 2 3 4 5 6 7 8 9	digits
	space (blank)

Special Characters	Meaning
	'*Horizontal Tabulation*'
+	plus sign
-	minus sign (hyphen)
*****	asterisk
/	slante (solidus)
=	equal sign
$	currency sign
,	comma
;	semi-colon
.	period (decimal point)
"	quotation mark
'	*apostrophe*
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol
:	colon
_	*underscore*

Punctuation Characters

Character	Meaning
,	comma
;	semi-colon
.	period
"	quotation mark
'	*apostrophe*
(	left parenthesis
)	right parenthesis
	space
	'*Horizontal Tabulation*'
=	equal sign
:	colon

COBOL Characters (cont)

Editing Characters

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)

Relation Characters

Character	Meaning
>	greater than
<	less than
=	equal to

The currency symbol will vary from country to country according to the customer's specification. The currency symbol (cs) denotes a single character position that corresponds to the hexadecimal 5B character position in the EBCDIC collating sequence; in this manual, it is represented in EBCDIC by the $ sign (see Appendix B).

The currency symbol may be altered through the use of the CURRENCY SIGN IS literal clause in the SPECIAL NAMES paragraph of the Environment Division (see Chapter 7). If the CURRENCY SIGN IS clause is not specified in the program, the default condition is CURRENCY SIGN IS cs, where cs is defined as above.

The comma character represents a decimal point when the DECIMAL-POINT IS COMMA clause is used in the SPECIAL-NAMES paragraph of the Environment Division (see Chapter 7).

3.2 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.

3.2.1 Separators

A separator is a character or two contiguous characters formed according to the following rules:

1. The punctuation characters space and |'Horizontal Tabulation'| are separators. Anywhere a space |or an 'Horizontal Tabulation'| is used as a separator or a part of a separator, more than one space |and/or 'Horizontal Tabulation'| may be used. All spaces |and/or 'Horizontal Tabulation'|following the separators comma, semi-colon, or period are considered part of that separator and are not considered to be the separator space |or 'Horizontal Tabulation'.|

2. Except when the comma is used in a PICTURE character-string, the punctuation characters comma and semi-colon, immediately followed by a space |or an 'Horizontal Tabulation',| are separators that may be used anywhere the separator space |or 'Horizontal Tabulation'| is used. They may be used to improve program readability.

3. The punctuation character period, when followed by a space |or an 'Horizontal Tabulation'| is a separator. It must be used only to indicate the end of a sentence, or as shown in formats.

4. The punctuation characters right and left parenthesis are separators. Parentheses may appear only in balanced pairs of left and right parentheses delimiting subscripts, a list of function arguments, reference modifiers, arithmetic expressions, |boolean expressions,| or conditions.

5. The punctuation characters quotation mark |and apostrophe| are separators. An opening quotation mark |or apostrophe| must be immediately preceded by a space, |'Horizontal Tabulation'| or left parenthesis; a closing quotation mark |or apostrophe, both| when paired with an opening quotation mark |or apostrophe, and when paired with the separator 'B"' or 'B",| must be immediately followed by one of the separators space, |'Horizontal Tabulation',| comma, semi-colon, period or right parenthesis.

6. Pseudo-text delimiters (two successive equal signs ==) are separators. An opening pseudo-text delimiter must be immediately preceded by a space |or an 'Horizontal Tabulation';| a closing pseudo-text delimiter must be immediately followed by one of the separator space, |'Horizontal Tabulation',| comma, semi-colon, or period.

   Pseudo-text delimiters may appear only in balanced pairs delimiting pseudo-text.

7. |The character 'B' immediately followed by the punctuation character quotation mark or apostrophe is a separator. This separator must be immediately preceded by a space or a left parenthesis.|

8. The punctuation character colon is a separator and is required when shown in the general formats.

9. The separators space |or 'Horizontal Tabulation'| may optionally immediately precede all separators except:

   As specified by reference format rules (see "Reference Format", Chapter 4).

   The separator closing quotation mark. In this case, a preceding space |or 'Horizontal Tabulation'| is considered as part of the non-numeric literal and not as a separator.

   The opening pseudo-text delimiter, where the preceding space |or 'Horizontal Tabulation'| is required.

10. The separators space |or 'Horizontal Tabulation'| may optionally immediately follow any separator except the opening quotation mark. In this case, a following space |or 'Horizontal Tabulation'| is considered as part of the non-numeric literal and not as a separator.

Any punctuation character which appears as part of the specification of a PICTURE character-string 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, |'Horizontal Tabulation',| comma, semi-colon, or period.

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

3.2.2 Character-Strings

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

3.2.2.1 COBOL Words

A COBOL word is a character-string of not more than 30 characters which forms a user-defined word, a system-name, a reserved word or a function-name. Each character of a COBOL word is selected from the set of letters, digits, the hyphen, and the character _ (underscore). The hyphen may not appear as the first or last character. Each lower-case letter is considered to be equivalent to its corresponding uppercase letter. Within a source program, reserved words and user-defined words form disjoint sets; reserved words and system-names form disjoint sets; function-names, system-names and user-defined words form intersecting sets. The same COBOL word may be used as a function-name, a system-name and as a user-defined word within a source program; and the class of a specific occurrence of this COBOL word is determined by the context of the clause or phrase in which it occurs.

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 letters A through Z (uppercase), |the letters a through z (lower-case),| the digits 0 through 9, the character - (hyphen) |and the character _ (underscore)| , except that the hyphen may not appear as the first or last character |and that the underscore may appear in library-names or text-names or in other types of user-defined words only if the word commences with "H_" or "h_" (underlining suppressed on "H_" and "h_" to make the underscore characters visible)|.

The types of user-defined words are:

1. alphabet-name
2. cd-name
3. class-name
4. condition-name
5. data-name
6. file-name
7. index-name
8. level-number
9. library-name
10. mnemonic-name
11. paragraph-name
12. program-name
13. record-name
14. report-name
15. section-name
16. segment-number
17. symbolic_character
18. text-name

Within a given source program, but excluding any contained program, the user-defined words are grouped into the following disjoint sets:

1. alphabet-names
2. cd-names
3. class-names
4. condition-names, data-names, and record-names
5. file-names
6. index-names
7. library-names
8. mnemonic-names
9. paragraph-names
10. program-names
11. report-names
12. section-names
13. symbolic_characters
14. 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, except as specified in the rules for uniqueness of reference (see "Uniqueness of Reference", this chapter).

With the exception of section-names, paragraph-names, segment-number, and level-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.

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 external switches.

A condition-name is used 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. A condition-name is also used in a SET statement, indicating that the associated value is to be moved to the conditional variable.

MNEMONIC-NAME: A mnemonic-name assigns a user-defined word to a hardware or operating system feature. These associations are established in the SPECIAL-NAMES paragraph of the Environment Division (see the "SPECIAL-NAMES Paragraph", Chapter 7).

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.

System-Names

A system-name is a COBOL word which is used to communicate with the operating environment.

There are three types of system-name:

1. computer-name (e.g. DPS7 ...)

2. input-output technique (e.g. UNBANNERED ...)

3. a specific feature of the hardware/software environment (e.g. SYSIN, SYSOUT, SWITCH-1 ...)

Reserved Words

A reserved word is a COBOL word |whose spelling (independent of whether it is written in upper case or lower case), is the same as a word| that is one of a specified list of words (see Appendix A). It may be used in COBOL source programs, but it must not appear in the programs as user-defined words. Reserved words can only be used as specified in the general formats.

There are three types of reserved words:

1. Required Words

2. Optional Words

3. Special Purpose Words.

REQUIRED WORDS: A required word is a word whose presence is required when the format in which the word appears is used in a source program.

Required words are of two types:

1. Key Words. Within each format, such words are uppercase and underlined.

2. Special Character Words. These are the arithmetic operators and relation characters.

OPTIONAL WORDS: Within each format, uppercase words that are not underlined are called optional words and may be specified at the user's option with no effect on the semantics of the format.

SPECIAL PURPOSE WORDS: There are two types of special purpose words:

1. Special Registers

2. Figurative Constants.

Special Registers: Certain reserved words are used to name and reference special registers. Special registers are certain compiler-generated storage areas whose primary use is to store information produced in conjunction with the use of specific COBOL features. Unless specified otherwise in these specifications, one special register of each type is allocated for each program. In the general formats of this specification, a special register may be used, unless otherwise restricted, wherever data-name or identifier is specified provided that the special register is the same category as the data-name or identifier. If qualification is allowed, special registers may be qualified as necessary to provide uniqueness (See "Qualification", this chapter).

1. |TALLY

   The reserved word TALLY is the name for a special register used in conjunction with the EXAMINE statement (see Chapter 11). The primary use of the TALLY register is to hold information produced by the EXAMINE statement. The word TALLY may also be used wherever an integer elementary data item may be used. TALLY is always defined as an unsigned integer whose PICTURE is 9(5).|

2. LINE-COUNTER

   The reserved word LINE-COUNTER is a name for a line counter that is generated for each Report Description entry in the Report Section of the Data Division. The implicit description is that of an unsigned integer that must be capable of representing a range of values from 0 through 999999. The value in LINE-COUNTER is maintained by the Report Writer Control System (RWCS), and is used to determine the vertical positioning of a report. LINE-COUNTER may be referenced only in the SOURCE clause of the Report Section and in Procedure Division statements; however, only the Report Writer Control System (RWCS) may change the value of LINE-COUNTER (See "LINE-COUNTER Rules", Chapter 8).

3. PAGE-COUNTER

   The reserved word PAGE-COUNTER is a name for a page counter that is generated for each Report Description entry in the Report Section of the Data Division. The implicit description is that of an unsigned integer that must be capable of representing a range of values from 1 through 999999. The value in PAGE-COUNTER is maintained by the Report Writer Control System (RWCS) and is used by the program to number the pages of a report. PAGE-COUNTER may be referenced only in the SOURCE clause of the Report Section and in Procedure Division statements (See "PAGE-COUNTER Rules", Chapter 8).

4. LINAGE-COUNTER

   The reserved word LINAGE-COUNTER is a name for a line counter generated by the presence of a LINAGE clause in a File Description entry. |LINAGE-COUNTER is always defined as an unsigned integer whose PICTURE is 9(6).| LINAGE-COUNTER may be referenced only in Procedure Division statements; however, only the input-output control system may change the value of LINAGE-COUNTER.

5. DEBUG-ITEM

The reserved word DEBUG-ITEM is the name for a special register generated automatically 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. (See "The Debugging Facility", Chapter 16).

|6. LENGTH OF data-name

The reserved words LENGTH OF followed by a data-name are the qualified name for a special register that is generated to contain the number of character positions that the data item referenced by data-name contains. The implicit description is that of an unsigned integer whose size is the number of characters necessary to contain its value. This register may only be referenced by Procedure Division statements where a numeric literal is allowed. Every data-name may be qualified. If data-name is USAGE BIT, it must be aligned. If the data description of data-name contains an occurs clause, it must be subscripted; the DEPENDING ON clause, if any is ignored.|

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

Function-Names

A function-name is a word that is one of a specified list of words which may be used in COBOL source programs. The same word, in a different context, may appear in a program as a user-defined word or a system-name.

3.2.2.2 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 belong to one of the types boolean, non-numeric or numeric.

|Boolean Literals

A boolean literal is a character-string delimited on the left by the separator 'B"' and on the right by the quotation mark separator. The character-string consists only of boolean characters. The value of a boolean literal is the string of boolean characters itself, excluding the delimiting separators. All boolean literals are of the category boolean (See the "PICTURE clause", Chapter 9).

Boolean literals of 1 through 256 boolean characters in length are allowed by the compiler.

The separator 'B" and the apostrophe may be used as boolean literal delimiters instead of 'B"' and the quotation mark provided that the first non-numeric or boolean literal (whichever is encountered first) is delimited by an apostrophe or 'B". In this case, all boolean literals must be delimited by 'B" and the apostrophe.|

Non-numeric Literals

A non-numeric literal is a character-string delimited at the beginning and at the end by the separator quotation mark. The string of characters may include any character in the computer's character set, |some or all of which may be symbolic-character-string.|

|Non-numeric literals of 1 through 256 characters in length are allowed by the compiler.|

American National Standard COBOL allows for non-numeric literals of 1 through 160 characters in length.

FORMAT:

```
  { character-1                          }
" { |--------------------------------| }... "
  { |  " {symbolic-character-1}... " | }
    |--------------------------------|
```

SYNTAX RULES:

1. Character-1 may be any character in the computer character set.

2. If character-1 is to represent the quotation mark, two contiguous quotation mark characters must be used to represent a single occurrence of that character.

|3. Symbolic-character-1 must be formed from the character set '0', '1', ..., '9'.

4. Symbolic-character-1 must not contain more than thirty (30) characters.

5. If more than one symbolic-character-1 is specified, the separator comma or space must appear between two consecutive occurrences of symbolic-character-1.

6. The first or the only symbolic-character-1 of a series must be immediately preceded by a quotation mark.|

GENERAL RULES:

1. The value of a non-numeric literal in the object program is the value represented by character-1 |or the character represented by symbolic-character-1.

2. The separator comma or space that appears between two consecutive occurrences of symbolic-character-1 is not part of the value of the non-numeric literal.

3. The punctuation quotation mark that delimits the symbolic-character-string is not part of the value of the non-numeric literal.|

4. The separator quotation mark that delimits the non-numeric literal is not part of the value of the non-numeric literal.

5. All other punctuation characters that appear in the non-numeric literal are part of the value of the non-numeric literal.

6. All non-numeric literals are of category alphanumeric.

|7. Symbolic-character-1 must be a numeric value that specifies the ordinal number of a character within EBCDIC character set. At object time, each symbolic-character in the non-numeric literal is replaced by the character from the native character set it represents.

8. A symbolic-character in a non-numeric literal occupies one character-position.

9. The apostrophe may be used as a non-numeric literal delimiter instead of the quotation mark provided that the first non-numeric or boolean literal (whichever is encountered first) is delimited by an apostrophe or 'B". In this case, all non-numeric literals must be delimited by apostrophes, and apostrophes within non-numeric literals are represented by 2 contiguous apostrophes.|

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, the character 'E', and/or the decimal point.

|There are two representations for a numeric literal, fixed-point and floating-point.|

The rules for the formation of |fixed-point| numeric literals are as follows:

1. A literal must contain at least 1 digit but no more |than 30 digits.|

   American National Standard COBOL allows for literals of 1 through 18 characters in length.

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, it is an integer.

   A literal that conforms to the rules for the formation of |fixed-point| numeric literals but is enclosed in quotation marks, is a non-numeric literal and is treated as such by the compiler.

4. The value of a |fixed-point| numeric literal is the algebraic quantity represented by the characters in the literal. Every |fixed-point| numeric literal is category numeric (see the "PICTURE" clause, Chapter 9). The size of a numeric literal in Standard Data Format characters is equal to the number of digits in the string of characters as specified by the user.

|The rules for the formation of floating-point numeric literals are as follows:

1. A floating-point numeric literal must contain at least five and no more than 36 characters and be in the form:|

```
[ + ]       { + }
[   ] k.mE  {   } n
[ - ]       { - }
```

   |Where:

   The brackets indicate that the selection of a positive or negative sign is optional.

   The symbols 'k' and 'm' represent the significand. Each represents zero or more digits. The significand must contain at least one digit and no more than 30.

   The symbol 'E' is a required character that separates the exrad from the significand in floating-point notation.

   The symbol 'n' represents the digits of the exrad. The exrad must contain at least one and no more than two digits.

2. The significand may be signed. If a sign is used, it must appear as the leftmost character of the significand. If the significand is unsigned, the floating-point literal is positive. The exrad must be signed.

3. The significand must contain a decimal point. The exrad must be an integer.

4. The value of a floating-point literal is the product of the value of its significand and the quantity derived by raising ten to the power indicated by the exrad.

5. If all the digits in the significand are zero: the sign of the significand, if present, must be positive; the sign of the exrad must be positive; and all the digits of the exrad must be zero.|

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.

Figurative constant values and the reserved words used to reference them are as follows:

1. `[ALL] ZERO, [ALL] ZEROS, [ALL] ZEROES`

   Represents the numeric value '0', |or one or more of the boolean character '0',|or one or more of the character '0' from the computer's character set.

2. `[ALL] SPACE, [ALL] SPACES`

   Represents one or more of the character space from the computer's character set.

3. `[ALL] HIGH-VALUE, [ALL] HIGH-VALUES`

   Except in the ALPHABET |or alphabet-name| clause, represents one or more of the character that has the highest ordinal position in the program collating sequence.

4. `[ALL] LOW-VALUE, [ALL] LOW-VALUES`

   Except in the ALPHABET |or alphabet-name| clause, represents one or more of the character that has the lowest ordinal position in the program collating sequence.

5. `[ALL] QUOTE, [ALL] 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 non-numeric literal. Thus, QUOTE ABD QUOTE is incorrect as a way of stating the non-numeric literal |"ABD". If the apostrophe is the non-numeric literal delimiter, the figurative constant QUOTE represents one or more of the character "' (apostrophe).|

6. `[ALL] literal`

   Represents all or part of the string generated by successive concatenations of the characters comprising the literal. The literal must be |either a boolean literal or| a non-numeric literal. The literal must not be a figurative constant.

7. `[ALL] symbolic-character`

   Represents one or more of the character specified as the value of this symbolic-character in the SYMBOLIC CHARACTERS clause of the SPECIAL-NAMES paragraph (See the "SPECIAL-NAMES Paragraph", Chapter 7).

When a figurative constant represents a string of one or more characters, the compiler determines the length of the string from the context in which it appears in the program, according to the following rules:

1. When a figurative constant is specified in a VALUE clause, or when a figurative constant is associated with another data item (e.g., 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 greater than or equal to the number of character positions in the associated data item. This resultant string is then truncated from the right until it is equal to the number of character positions in 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, other than ALL literal, is not associated with another data item (as when the figurative constant appears in a DISPLAY, STOP, STRING, or UNSTRING statement), the length of the string is one character.

3. When the figurative constant ALL literal is not associated with another data item, the length of the string is the length of the literal.

A figurative constant may be used whenever 'literal' appears in a format with the following exceptions:

1. If the literal is restricted to a numeric literal, the only figurative constant permitted is ZERO (ZEROS, ZEROES).

|2. If the literal is restricted to a boolean literal, the only figurative constants permitted are ZERO (ZEROS, ZEROES) and ALL literal.|

3. Associating the figurative constant ALL literal where the length of the literal is greater than one with a data item that is numeric or numeric edited is an obsolete feature in the current revision of American National Standard COBOL. This obsolete feature is to be deleted from the next revision of American National Standard COBOL.

4. When a figurative constant other than ALL literal is used, the word ALL is redundant and is used for readability only.

Except in the ALPHABET |or alphabet-name| clause, when the figurative constants HIGH-VALUE(S) or LOW-VALUE(S) are used in the source program, the actual characters associated with each figurative constant depend upon the program collating sequence specified (see "OBJECT-COMPUTER Paragraph" and "SPECIAL-NAMES Paragraph" in the Environment Division, Chapter 7).

Each reserved word which is used to reference a figurative constant value is a distinct character-string with the exception of the constructs using the word ALL, such as ALL literal, ALL SPACES, etc., which are composed of two distinct character-strings.

3.2.2.3 Picture Character-Strings

A PICTURE character-string consists of certain symbols which are composed of the currency symbol and certain combinations of characters in the COBOL character set. An explanation of the PICTURE character-string and the rules that govern its use are given under the appropriate paragraph (See the "PICTURE clause", Chapter 9).

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.

3.2.2.4 Comment-Entries

A comment-entry is any entry in the Identification Division that may be any combination of characters from the computer's character set. Comment-entry is an obsolete element in the current revision of American National Standard COBOL because it is to be deleted from the next revision of American National Standard COBOL.

3.3 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 all characters of the COBOL character set to describe non-numeric data items.

3.3.1 Logical Record Concept

In order to separate the logical characteristics of data from the physical characteristics of the data storage media, separate clauses or phases are used. The following paragraphs discuss the characteristics of files.

3.3.1.1 Physical Aspects of a File

The physical aspects of a file describe the data as it appears on the input or output media and include such features as:

1. The mode in which the data file is recorded on the external medium.
2. The grouping of logical records within the physical limitations of the file medium.
3. The means by which the file can be identified.

3.3.1.2 Conceptual Characteristics of a File

The conceptual characteristics of a file are the explicit definition of each logical entity within the file itself. In a COBOL program, the input or output statements refer to one logical record.

It is important to distinguish between a physical record and a logical record. A COBOL logical record is a group of related information, uniquely identifiable and treated as a unit.

A physical record is a physical unit of information whose size and recording mode is convenient to a particular computer for the storage of data on an input or output device. The size of a physical record is hardware dependent and bears no direct relationship to the size of information contained on a device.

A logical record may be contained within a single physical unit; or several logical records may be contained within a single physical unit; or a logical record may require more than one physical unit to contain it. There are several source language methods available for describing the relationship of logical records and physical units. When a permissible relationship has been established, control of the accessibility of logical records as related to the physical unit must be provided by the interaction of the object program on the hardware and/or software system. In this manual, references to records means to logical records, unless the term "physical record" is specifically used.

The concept of a logical record is not restricted to file data but is carried over into the definition of working storage. Thus, working storage may be grouped into logical records and defined by a series of Record Description entries.

When a logical record is transferred to or from a physical unit, any translation required by the presence of a CODE-SET clause is accomplished. Padding characters are added or deleted as necessary. None of the clauses used to describe the data in the logical record have any effect on this transfer.

3.3.1.3 Record Concepts

The 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.

3.3.2 Concepts 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 reference.

Subdivisions of a Record

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. There are special level-numbers 66, 77 and 88, which are exceptions to this rule (see below). Separate entries are written in the source program for each level-number used.

A group includes all group and elementary items following it until a level-number less than or equal to the level-number of that group is encountered. All items which are immediately subordinate to a given group item must be described using identical level-numbers greater than the level-number used to describe that group item.

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

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

2. Entries that specify non-contiguous working-storage and linkage data items.

3. Entries that specify condition-names.

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

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

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

3.3.3 Concept of Classes of Data

The categories of data items are alphabetic, |boolean,| numeric, numeric edited, alphanumeric, and alphanumeric edited (see the "PICTURE clause", Chapter 9). These are grouped into the classes: alphabetic, |boolean,| numeric, and alphanumeric. For alphabetic, |boolean,| 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 |or a pointer| belongs to one of the classes and further 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.

Every data item which is a function is an elementary item, and belongs to one of the categories alphanumeric or numeric, and to the corresponding class; the category of each function is determined by the definition of the function. This definition is made in these specifications.

The following table depicts the relationship of the class and categories of data items.

Table 3-2. Data Item Class and Category

LEVEL OF ITEM	CLASS	CATEGORY
Elementary	Alphabetic	Alphabetic
	Boolean	Boolean
	Numeric	Numeric
	Alphanumeric	Numeric Edited
		Alphanumeric Edited
		Alphanumeric
Non Elementary (Group)	Alphanumeric	Alphabetic
		Boolean
		Numeric
		Numeric Edited
		Alphanumeric Edited
		Alphanumeric

3.3.4 Selection of Character Representation and Radix

The value of a numeric item may be represented in either binary or decimal form. In addition there are several ways of expressing decimal. The selection of radix is dependent upon factors included in such clauses as USAGE (see Chapter 9).

3.3.4.1 Size of an Elementary Item

The size of an elementary data item or a group item is the number of characters in Standard Data Format of the item. Synchronization and usage may cause a difference between this size and the actual number of characters required for internal representation

3.3.4.2 Data Types

The types of data supported by GCOS COBOL are listed in the table "Data Representation in the DPS7 System" below, according to factors included in the USAGE and PICTURE clauses. The usage of an item specifies the format of the data item in computer storage (see "USAGE", Chapter 9). The following paragraphs describe how each data type is represented in internal memory.

DISPLAY Data Item

Character-strings are defined, explicitly or implicitly, by a USAGE IS DISPLAY clause. Character-strings, represented in EBCDIC code, are stored in memory in contiguous bytes with one character per byte. These character-strings may be non-numeric data as well as decimal numbers.

Packed Decimal Number

A packed decimal number (USAGE IS COMP, PACKED DECIMAL, COMP-3 or COMP-8) is represented as a series of contiguous bytes, each containing two 4-bit digit encoding portions, except for the rightmost byte. The leftmost four bits of this byte represent a digit, while the rightmost four bits define a sign except as noted below (Algebraic Signs). However, if the USAGE IS COMP |or COMP-3,| and the PICTURE character-string does not show a sign, the rightmost four bits represent the rightmost digit. In a usage COMP-5 data item, the sign has the ASCII representation (see the "USAGE clause", Chapter 9).

Usage BINARY Fixed-Point Data

A Usage BINARY Fixed Point number is represented as a 16-bit, 32-bit, 48-bit or 64-bit integer depending on the number of decimal positions specified in the PICTURE clause. |Differing in that from the COMP-1 and COMP-2 usages,| the characteristics specified in the PICTURE clause for a usage BINARY data item are significant.

Usage BIT Data Item

A usage BIT data item is represented as a series of consecutive bits, one bit per character position as specified in the PICTURE clause. When the SYNCHRONIZED clause is not present in a usage BIT data item description, this data item is not necessarily aligned on a byte boundary.

Fixed Binary Data

Fixed binary data can be specified as either 16-bit binary (USAGE IS COMP-1 and no PICTURE, or a PICTURE showing less than 5 digits) or 32 bit binary (USAGE IS COMP-2 or COMP-1 with a PICTURE showing more than 4 digits). The short binary data item consists of two contiguous bytes; the long binary data item, of four contiguous bytes. In both types of data, a decimal point is assumed to be to the right of the least significant bit. Negative values are stored in two's complement form.

Floating Binary Data

Floating binary data can be specified either as 32-bit binary (USAGE IS COMP-9), or 64-bit binary (USAGE IS COMP-10), or 128-bit binary (USAGE IS COMP-15). The short floating binary data item gives Single Precision (a precision of approximately 7 decimal digits). The long floating binary data item gives Double Precision (a precision of approximately 16 decimal digits). The extended floating point binary data item gives Quadruple Precision (a precision of approximately 27 decimal digits).|

Index Data Item

An index data item (USAGE IS INDEX) consists of 48 bits (six bytes). (See "Indexing", later in this chapter, for a description of its use.)

|Pointer Data Item

A pointer data item (USAGE IS POINTER) is a 32 bit direct ITS (four bytes) data item.|

Table 3-3. Data Representation in the DPS 7 System

USAGE	Machine Description	PICTURE
DISPLAY	EBCDIC byte	R
* COMPUTATIONAL or COMP	Packed decimal (possibly without sign position depending on PICTURE)	R
PACKED DECIMAL	Packed decimal (always with sign position)	R
BINARY	16, 32, 48 or 64 bit fixed binary	R
COMPUTATIONAL-1 or COMP-1	16-bit fixed binary (possibly 32-bit fixed binary depending on PICTURE)	NR
COMPUTATIONAL-2 or COMP-2	32-bit fixed binary	NR
* COMPUTATIONAL-3 or COMP-3	Packed decimal (possibly without sign position depending on PICTURE)	R
COMPUTATIONAL-5 or COMP-5	Packed decimal (always with sign position, sign has ASCII representation)	R
COMPUTATIONAL-8 or COMP-8	Packed decimal (always with sign position)	R
COMPUTATIONAL-9 or COMP-9	Floating binary single precision	NA
COMPUTATIONAL-10 or COMP-10	Floating binary double precision	NA
COMPUTATIONAL-15 or COMP-15	Floating binary quadruple precision	NA
BIT	1 bit per character position	R
POINTER	32 bit direct ITS	NA
INDEX	6 bytes	NA

NOTES: R = PICTURE clause required in data description entry

NR = PICTURE clause not required in data description entry

NA = PICTURE clause not allowed in data description entry.

|* These items have the same meaning, unless specified otherwise in the Default Section of the Control Division.|

3.3.5 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, for example, 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 for DISPLAY data items. This clause is optional; if it is not used, the operational sign is overpunch in the trailing position.

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

3.3.6 Standard Rules for Data Alignment

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

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

   a. For fixed-point numeric items, the actual radix point of the value of the sending data item is aligned with the radix point of the receiving item and the data is moved to the receiving digit positions with zero fill or truncation on either end as required. When a radix point is not explicitly specified, the data item is treated as if it had an assumed radix point immediately following its rightmost digit.

   |b. For floating-point numeric items, the sending field is normalized to remove leading zero digits. That is, the most significant non-zero digit of the value of the sending item is aligned at the leftmost digit position of the significand of the receiving item and the data is moved to the receiving digit positions with zero fill or truncation on the right as required. The exrad of the receiving data item is adjusted by a value equal to the number of digit positions between the most significant non-zero digit of the sending item and its radix point. If the most significant non-zero digit is to the left of the radix point, the adjustment is positive; if it is to the right of the radix point, the adjustment is negative.

   If the absolute value of the exrad, as adjusted, is larger than the maximum number that can be accommodated by the exrad of the receiving data item, the results of the operation are undefined.|

2. When the receiving data item is described as numeric edited:

   a. For a fixed-point numeric edited data item, the actual radix point of the value of the sending data item is aligned with the decimal point of the receiving item and the

data is moved 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. When a decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately following its rightmost digit.

|b. For a floating-point numeric edited data item, the most significant non-zero digit of the value of the sending item is aligned at the leftmost digit position of the significand of the receiving item and the data is moved with zero fill or truncation on the right, as required. The adjusted exrad value is aligned at the rightmost digit position of the exrad of the receiving data item and is zero filled on the left, as required. If the absolute value of the exrad, as adjusted, is larger than the maximum number that can be accommodated by the exrad of the receiving data item, the results of the operation are undefined.|

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.

|4. If the receiving data item is boolean, the sending data is moved to the receiving boolean positions and aligned at the leftmost boolean position in the data item with boolean character zero 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 under "JUSTIFIED", Chapter 9.

3.3.7 Data Allocation

The natural addressing boundaries for data are the byte, half-word (two bytes), word (four bytes), and double-word (eight bytes). While data may be stored in memory during program execution without regard for these boundaries, alignment of data on natural boundaries reduces the access time for accessing and storage of data. Use of the SYNCHRONIZED clause to allocate data on natural boundaries enhances the speed of processing at the expense of efficient utilization of space, of easy communication between programs, and of easy exchange of programs.

The description of data allocation is given in three parts:

- Alignment
- Unused space
- Allocation

The allocation rules apply to any 01-level group item with no REDEFINES clause or any group item with a REDEFINES clause independently of any of its subordinate items described with a REDEFINES clause and items subordinated to those items.

3.3.7.1 Alignment

The rules for alignment in the order of application are:

1. All level 77 and elementary level 01 index data items begin on a byte boundary, other level 77 and elementary level 01 data items begin on the synchronized boundary as described below in SYNCHRONIZATION BOUNDARIES.

2. All level 01 group data items begin on a word boundary, |or a double word boundary if the record contains a COMP-10 OR COMP-15 item,| including in the LINKAGE SECTION. Therefore the user must ensure that the operands referenced in any USING clause within a CALL statement are properly aligned.

3. SYNC alignment is described below in "Synchronization Boundaries".

4. The default alignment for all data types, |other than USAGE BIT boolean,| is byte if SYNC is not specified.

|5. The default alignment for USAGE BIT boolean is bit if SYNC is not specified.|

3.3.7.2 Unused Space

In addition to the space allocated for an elementary data item, unused space (called "FILLER") may be allocated in the following situations:

1. Following each occurrence, including the last, of a repeating group data item, in order to align the next occurrence of that data item on its required boundary. This is called a "type 1 FILLER".

2. Following a data item (elementary or group), or following the last occurrence of a repeating data item, in order to align the next data item on its required boundary, |or following the last elementary item in a group in order to terminate this group on a byte boundary.| This is called a "type 2 FILLER".

   Such a FILLER is in addition to any type 1 FILLER which follows the last occurrence of a repeating data item.

|3. Preceding a SYNCHRONIZED RIGHT elementary data item, or following a SYNCHRONIZED LEFT data item, when the size of this elementary item is less than the size of the memory portion comprised between the leftmost natural boundary and the rightmost natural boundary which define this item (e.g. a SYNCHRONIZED USAGE BIT boolean item whose size is not a multiple of eight). This is called a "type 3 FILLER". Such a FILLER is in addition to any necessary preceding type 2 or type 3 FILLER.|

When the SYNCHRONIZED clause is specified for an elementary item which contains a REDEFINES clause or which is subordinate to an item which contains a REDEFINES clause, the elementary item must not require the addition of any type 1 or type 2 unused bytes. The user must provide the necessary FILLER to align this item on the proper boundary.

3.3.7.3 Allocation

The rules for allocation are:

1. Elementary items are allocated in source language order from the first data item in the record (or the redefining group) to the last. The first elementary item in a record is allocated on a word boundary |or a double word boundary if the record contains a COMP-10 OR COMP-15 item.| The first elementary item in a redefining group is given the same address as the redefined item.

2. If previously allocated space does not end on the required boundary for alignment of the current elementary item, then unused space is allocated until the boundary requirement is satisfied (type 2 FILLER). |If the current elementary item is the first elementary item subordinate to a group, the required boundary is at least byte boundary.|

3. For a non repeating elementary item, space is allocated according the rules given below in "Size of Elementary Items".

4. For a repeating elementary item, the space allocated is the product of the space allocated for one such item (see "Size of Elementary Items" below) and the number of occurrences.

5. The space allocated for a non repeating group is the sum of the spaces allocated to its direct subordinate items. |If necessary, a type 2 FILLER is included at the end of

the group to extend its size to the full last allocated byte.| If the first direct subordinate item is SYNCHRONIZED and if it requires the allocation of a type 2 FILLER, this type 2 FILLER is not included in the space allocated for the group. Any other type 2 FILLER |and any type 3 FILLER| due to the subordinate items are included in the space allocated for the group.

6. The space allocated for a repeating group is determined as follows:

   The space allocated for one occurrence of the group as defined in rule 5 is increased to include the type 1 FILLER necessary to align the next occurrence the same way the first occurrence was aligned with respect to the most stringent boundaries required for the subordinate SYNCHRONIZED items.

   In other terms:

   - |If the repeating group contains no SYNCHRONIZED COMP-1, SYNCHRONIZED COMP-2, SYNCHRONIZED COMP-9, SYNCHRONIZED COMP-10 nor SYNCHRONIZED COMP-15 item, no FILLER is allocated
   - If the repeating group contains SYNCHRONIZED COMP-1 with no PICTURE or with a PICTURE showing less than 5 digit positions items but no SYNCHRONIZED COMP-2, COMP-9, COMP-10 or COMP-15 items, its size is rounded to a multiple of 2
   - If the repeating group contains SYNCHRONIZED POINTER, SYNCHRONIZED COMP-1 with a PICTURE showing more than 4 digit positions, SYNCHRONIZED COMP-2 or SYNCHRONIZED COMP-9 items but no SYNCHRONIZED COMP-10 and no SYNCHRONIZED COMP-15 items, its size is rounded to a multiple of 4.
   - If the repeating group contains SYNCHRONIZED COMP-10 or SYNCHRONIZED COMP-15 items, its size is rounded to a multiple of 8.|

The allocated space for the entire group data item is then the product of the space allocated for one occurrence and the number of occurrences.

3.3.7.4 Size of Elementary Items

The size of an elementary item depends on its USAGE, its PICTURE and its SIGN clauses.

The size of a USAGE DISPLAY item is the number of characters specified in its PICTURE clause. When the first character in the PICTURE character-string is "S", this symbol is counted in the item size only if the applicable SIGN clause contains the SEPARATE phrase.

The size in bytes of a USAGE PACKED DECIMAL item is the integer part of the quotient of n+2 divided by 2, where "n" is the number of digits specified by the PICTURE clause.

The size of a USAGE BINARY item is 2 bytes if its PICTURE describes less than 5 digits, 4 bytes if it describes 5 to 9 digits, 6 bytes if it describes 10 to 14 digits, 8 bytes if it describes more than 14 digits.

|The size of a USAGE COMP-1 item is 2 bytes if it has no PICTURE or if its PICTURE describes less than 5 digits. Otherwise, it is 4 bytes.

The size of a USAGE COMP-2 item is 4 bytes.

The size in bytes of a USAGE COMP-3 OR COMP-5 item is the integer part of the quotient of n+s+1 divided by 2, where "n" is the number of digits specified by the PICTURE clause, "s" is 1 if the PICTURE string contains an "S", and "s" is 0 (zero) if the PICTURE string contains no "S".

The size in bytes of a USAGE COMP-8 item is the integer part of the quotient of n+2 divided by 2, where "n" is the number of digits specified by the PICTURE clause.

The size of a USAGE COMP-9 item is 4 bytes.

The size of a USAGE COMP-10 item is 8 bytes.

The size of a USAGE COMP-15 item is 16 bytes.|

The size of a USAGE INDEX item is 6 bytes.

|The size of a USAGE BIT item is the number of boolean positions specified in its PICTURE clause. If the USAGE BIT item's description contains the SYNCHRONIZED clause, the space allocated for it includes any necessary type 3 FILLER.

The size of a USAGE POINTER item is 4 bytes.|

3.3.7.5 Synchronization of Boundaries

The alignment of a SYNCHRONIZED item depends on its USAGE as shown below:

Table 3-4. Boundary Requirements for Synchronized Data

Usage	Synchronized Boundary
BINARY	Half-word or word
BIT	Byte
COMP	Byte
COMP-1	Half-word or word
COMP-2	Word
COMP-3	Byte
COMP-5	Byte
COMP-8	Byte
COMP-9	Word
COMP-10	Double-word
COMP-15	Double-word
DISPLAY	Byte
INDEX	Not applicable
PACKED DECIMAL	Byte
POINTER	Word

Examples (Type 1 FILLER)

Example 1:

```
01 A.
02 B PIC XXX.
02 C OCCURS 2.
03 D PIC X.
03 E SYNC COMP-2.
```

The following equivalent declaration shows the type 1 FILLER inserted by the compiler:

```
01 A.
02 B PIC XXX.
02 C OCCURS 2.
03 D PIC X.
03 E COMP-2.
03 FILLER PIC XXX.
```

The offset of E (I) must be a multiple of 4. The offset of C (1) is a multiple of 4 plus 3, this leads E(1) to be correctly aligned. In order to obtain next occurrences of E on the correct boundaries, all occurrences of C must have an offset in multiples of 4 plus 3 also. Thus the compiler inserted the PIC XXX FILLER to make the length of one occurrence of C be always a multiple of 4.

Example 2:

If C contained two COMP-1 items instead of a COMP-2 item, the size of an occurrence of C would need to be a multiple of 2 only. Then, a PIC X FILLER would be enough:

```
01 A.
02 B PIC XXX.
02 C OCCURS 2.
03 D PIC X.
03 E.
04 E1 COMP-1 SYNC.
04 E2 COMP-1 SYNC.
```

becomes

```
01 A.
02 B PIC XXX.
02 C OCCURS 2.
03 D PIC X.
03 E.
04 E1 COMP-1.
04 E2 COMP-1.
03 FILLER PIC X.
```

Example 3:

If the SYNCHRONIZED item is subordinate to more than one repeating group, the compiler may insert more than one type 1 FILLER:

```
01 A.
02 B PIC XXX.
02 C OCCURS 2.
03 D PIC X.
03 EF OCCURS 2.
04 E COMP-2 SYNC.
04 F PIC XX.
```

becomes

```
01 A.
02 B PIC XXX.
02 C OCCURS 2.
03 D PIC X.
03 EF OCCURS 2.
04 E COMP-2.
04 F PIC XX.
04 FILLER PIC XX.
03 FILLER PIC XXX.
```

Examples (Type 2 FILLER)

Example 1:

```
01 A.
02 B.
03 C PIC X.
03 D PIC XX.
02 E.
03 F COMP-1 SYNC.
03 G COMP-2 SYNC.
```

The following equivalent declaration shows two type 2 FILLER inserted by the compiler:

```
01 A.
02 B.
03 C PIC X.
03 D PIC XX.
02 FILLER PIC X.
02 E.
03 F COMP-1.
03 FILLER PIC XX.
03 G COMP-2.
```

The PIC XX FILLER due to G is included in E since G is subordinate to E and it is not the first element of E. The PIC X FILLER due to F is not included in E because F is the first element of E.

Example 2:

A 3-bit type 2 FILLER is inserted before the group C and a 6-bit type 2 FILLER is added at its end to make it start and end on byte boundaries. A 3-bit type 2 FILLER is added at the end of the record A to make it end on a byte boundary.

```
01 A BIT.
02 B PIC 1(5).
02 C.
03 D PIC 1(5).
03 E PIC 1(5).
02 F PIC 1(5).
```

becomes

```
01 A BIT.
02 B PIC 1(5).
02 FILLER PIC 111.
02 C.
03 D PIC 1(5).
03 E PIC 1(5).
03 FILLER PIC 1(6).
02 F PIC 1(5).
02 FILLER PIC 111.
```

Example (Type 3 FILLER)

Type 3 FILLER are inserted to align SYNCHRONIZED USAGE BIT items.

```
01 A BIT.
02 B SYNC LEFT PIC 1.
02 CSYNC RIGHT PIC 1.
```

becomes

```
01 A BIT.
02 B PIC 1.
02 FILLER PIC 1(7).
02 FILLER PIC 1(7).
02 C PIC 1.
```

Example (Redefining)

Re-definitions do not affect the allocation of the redefined data.

This is a combination of Type 1 FILLER examples 1 and 2.

```
01 A.
02 B PIC XXX.
02 C OCCURS 2.
03 D PIC X.
03 E.
04 E1 COMP-1 SYNC.
04 E2 COMP-1 SYNC.
03 EE REDEFINES E COMP-2 SYNC.
```

becomes

```
01 A.
02 B PIC XXX.
02 C OCCURS 2.
03 D PIC X.
03 E.
04 E1 COMP-1.
04 E2 COMP-1.
03 EE REDEFINES E COMP-2.
03 FILLER PIC X.
```

This is an error: the type 1 FILLER will make all occurrences of E1 and E2 be aligned on half words, but only those occurrences of EE which have an odd subscript will be word aligned.

3.3.8 Definition of a Legible Equivalent

The legible equivalent specifies the form of the data under which it is represented on an external medium, for the purpose of ACCEPTing it or DISPLAYing it. The specification is given in terms of conceptual data items that would be used as an editing/de-editing area between the program and the external device. The legible equivalent consists of one, two or three contiguous conceptual data items whose usage is DISPLAY, and whose description is deduced from the description of the data item of which it is a legible equivalent.

The legible equivalent rules as specified in the current paragraph do not apply when the WITH CONVERSION phrase is used in a DISPLAY statement. (See the "DISPLAY Statement", Chapter 11).

There are several legible equivalents:

- depending on whether it is the input (ACCEPT) or output (DISPLAY) form
- depending on the suffix appended to the hardware-name used to specify the device on which the data is ACCEPTed or DISPLAYed. The possible suffixes are:

  -0 to specify that there is no legible equivalent; data is transferred to or from the external medium without conversion except when the WITH CONVERSION phrase is used in a DISPLAY statement. (See the "DISPLAY Statement", Chapter 11.)

  -1 to specify that the legible equivalent is the DISPLAY representation of the data in memory, i.e. that no editing takes place; it is the same form on input and output.

  -2 to specify that the legible equivalent is the DISPLAY representation of the data in memory; in addition, for numeric data on output, sign and decimal point editing takes place, and on input, a rather flexible form is permitted.

  -X to specify that the legible equivalent is the hexadecimal representation of the value of the data in memory independent of its description in the program.

In all cases group items behave as elementary alphanumeric items.

The following describe the various legible equivalents.

3.3.8.1 Legible Input Equivalent

The legible input equivalent is the form of the data on the external medium from which it will be read by an ACCEPT statement.

If the hardware-name suffix is -X, the representation of the legible input equivalent of a data item is the hexadecimal representation of the bit-coded value of the data item in memory. The following does not apply when the suffix is -X.

The representation of the legible input equivalent of an elementary data item whose category is alphabetic, alphanumeric, alphanumeric-edited, or numeric-edited is exactly the same as that of the data item of which it is the legible input equivalent.

The representation of the legible input equivalent of an elementary data item whose category is boolean is exactly the same as that of the data item of which it is the legible input equivalent, except that if this data item is of usage BIT, the usage of the legible input equivalent is DISPLAY.

The representation of the legible input equivalent of a POINTER data item is always an 8 hexadecimal character-string.

The representation of the legible input equivalent of an elementary data item whose category is numeric depends upon the suffix associated to the hardware-name.

a) Hardware-name suffix is -1

   The legible input equivalent is described by a picture character-string deduced from the picture and/or usage of the accepting data, as shown in the table "Legible Equivalents of Elementary Numeric Data Items" below.

b) Hardware-name suffix is -2

   The legible input equivalent has the form of a COBOL numeric literal, including floating point numeric literals; it may contain blanks before and/or after the literal; the point or the comma may be used as decimal mark; if the form is that of a floating point literal, the "E" may be upper-case or lower-case. The size of the legible input equivalent is the number of characters of the data on the external medium.

Elementary index-data-items cannot be ACCEPTed and therefore they have no legible input equivalent.

Index-names are not identifiers and therefore cannot be ACCEPTed.

The representation of the legible input equivalent of a group item is the same as that of a re-definition of that group item as an elementary alphanumeric data-item whose length would be the same as that of the group.

3.3.8.2 Legible Output Equivalent

The legible output equivalent is the form of the data on the external medium upon which it will be written by a DISPLAY statement.

If the hardware-name suffix is -X, the representation of the legible output equivalent of a data item is the hexadecimal representation of the bit-coded value of the data item in memory. The following does not apply when the suffix is -X.

The representation of the legible output equivalent of an elementary data item whose category is alphabetic, alphanumeric, alphanumeric-edited, or numeric-edited is exactly the same as that of the data item of which it is the legible output equivalent.

The representation of the legible output equivalent of an elementary data item whose category is boolean is exactly the same as that of the data item of which it is the legible output equivalent, except that if this data item is of usage BIT, the usage of the legible output equivalent is DISPLAY.

The representation of the legible output equivalent of a POINTER data item is always an 8 hexadecimal character-string.

The representation of the legible output equivalent of an elementary data item whose category is numeric depends upon the suffix associated to the hardware-name. It is described by a picture character-string deduced from the picture and/or usage of the displayed data, as shown in the table "Legible Equivalents of Elementary Numeric Data Items" below. The representation of the decimal mark is that implied by the DECIMAL-POINT clause.

Elementary index-data-items cannot be DISPLAYed and therefore they have no legible output equivalent.

Index-names are not identifiers and therefore cannot be DISPLAYed.

The representation of the legible output equivalent of a group item s the same as that of a re-definition of that group item as an elementary alphanumeric data-item whose length would be the same as that of the group.

Table 3-5. Legible Equivalents of Elementary Numeric Data Items

Hardware-name suffix	-1	-2
Applicable statements	ACCEPT, DISPLAY	DISPLAY
DISPLAY		
SIGN TRAILING SEPARATE		
S9(p)	9(p)+	-(p)9
S9(p)V9(q)	9(p)V9(q)+	-(p)9.9(q)
S9(p)P(r)	9(p)P(r)+	-(p+r)9
SP(r)9(q)	P(r)9(q)+	-.9(r+q)
other		
S9(p)	+9(p)	-(p)9
S9(p)V9(q)	+9(p)V9(q)	-(p)9.9(q)
S9(p)P(r)	+9(p)P(r)	-(p+r)9
SP(r)9(q)	+P(r)9(q)	-.9(r+q)
S9(p)VES9(e)	+9(p)VF+9(e)	-9(p).e+9(e)
S9(p)V9(q)ES9(e)	+9(p)V9(q)F+9(e)	-9(p).9(q)E+9(e)
SV9(q)ES9(e)	+V9(q)F+9(e)	-.9(q)E+9(e)
9(p)	9(p)	Z(p-1)9
9(p)V9(q)	9(p)V9(q)	Z(p-&)9.9(q)
9(p)P(r)	9(p)P(r)	Z(p+r-1)9
P(r)9(q)	P(r)9(q)	.9(r+q)
9(p)VES9(e)	9(p)VF+9(e)	9(p).e+9(e)
9(p)V9(q)ES9(e)	9(p)V9(q)F+9(e)	9(p).9(q)E+9(e)
V9(q)ES9(e)	V9(q)F+9(e)	.9(q)E+9(e)
BINARY	same as DISPLAY	same as DISPLAY
PACKED-DECIMAL	same as DISPLAY	same as DISPLAY
COMP-1 no picture	+9(5)	-(5)9
[S]9(p) p<5	+9(5)	-(5)9
p>4	+9(10)	-(10)9
COMP-2	+9(10)	-5(10)9
COMP-3	same as DISPLAY	same as DISPLAY
COMP-5	same as DISPLAY	same as DISPLAY
COMP-8	same as DISPLAY	same as DISPLAY
COMP-9	+V9(7)F+99	-9.9(6)E+99
COMP-10	+V9(16)F+99	-9.9(15)E+99
COMP-15	+V9(27)F+99	-9.9(26)E+99

NOTE

|"E" and "F" in the description of the legible equivalent indicate that the picture character-string is that of the exrad of a floating point legible equivalent, whereas the picture character-string that precedes to the left is that of the significand. "E" will be present in the legible equivalent whereas "F" is shown here for documentation only but will not be present in the legible equivalent (i.e. the exrad representation will immediately follow the significand representation). On output, the first digit of the significand of a floating point legible equivalent will be a non-zero digit unless the data item has a value zero in which case all digits of the legible equivalent will be "0" and signs "+" or " " as implied by the picture of the legible equivalent.|

3.3.9 Uniqueness of Reference

Every user-defined name in a COBOL program is assigned, by the user, to name a resource which is to be used in solving a data processing problem (See "User-Defined Words", this chapter). In order to use a resource, a statement in a COBOL program must contain a reference which uniquely identifies that resource. In order to ensure uniqueness of reference, a user-defined name may be qualified, subscripted, or reference modified as described in the following paragraphs.

When the same name has been assigned in separate programs to two or more occurrences of a resource of a given type, and when qualification by itself does not allow the reference in one of those programs to differentiate between the two identically named resources, then certain conventions which limit the scope of names apply. These conventions ensure that the resource identified is that described in the program containing the reference (See "Scope of Names", this chapter).

Unless otherwise specified by the rules for a statement, any subscripting and reference modification are evaluated only once as the first operation of the execution of that statement.

3.3.9.1 Qualification

Every user-defined name explicitly referenced in a COBOL source program must be uniquely referenced because either:

1. No other name has the identical spelling and hyphenation.

2. It is unique within the context of a REDEFINES clause (see the "REDEFINES Clause", Chapter 8).

3. The name exists within a hierarchy of names such that reference to the name can be made unique by mentioning one or more of the higher-level names in the hierarchy.

   These higher-level names are called qualifiers and this process that specifies uniqueness is called qualification. Identical user-defined names may appear in a source program; however, uniqueness must then be established through qualification for each user-defined name explicitly referenced, except in the case of re-definition. All available qualifiers need not be specified so long as uniqueness is established. Reserved words naming the special registers require qualification to provided uniqueness of reference whenever a source program would result in more than one occurrence of any of these special registers. A paragraph-name or section-name appearing in a program may not be referenced from any other program, |however a global paragraph-name or section-name may be referenced in GO TO statements from programs contained (directly or not) in the program in which the global paragraph-name or section-name is declared.|

4. A program is contained within a program or contains another program (See "Scope of Names", this chapter).

Regardless of the above, the same data-name must not be used as the name of an external record and as the name of any other external data item described in any program contained within or containing the program which describes that external data record. The same data-name must not be used as the name of an item possessing the global attribute and as the name of any other data item described in the program which describes that global data item.

The general formats for qualification are:

Format 1

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

Format 2

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

Format 3

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

Format 4

```
                  { IN }
LINAGE-COUNTER    { OF }  file-name
```

Format 5

Format 6

```
{PAGE-COUNTER}   { IN }
{              } { OF }  report-name
{LINE-COUNTER}   { OF }

                  {IN}                  {IN}
                { {  } data-name-4   [{  } report-name ] }
                { {OF}                {OF}               }
data-name-3     {                                        }
                { {IN}                                   }
                { {  }  report-name                      }
                  {OF}
```

The rules for qualification are as follows:

1. For each non-unique user-defined name that is explicitly referenced, uniqueness must be established through a sequence of qualifiers which precludes any ambiguity of reference.

2. IN and OF are logically equivalent.

3. A name can be qualified even though it does not need qualifications; if there is more than one combination of qualifiers that ensures uniqueness, then any such set can be used.

4. In Format 1, each qualifier must be the name associated with a level indicator, the name of a group item to which the item being qualified is subordinate, or the name of the conditional variable with which the condition-name being qualified is associated. Qualifiers are specified in the order of successively more inclusive levels in the hierarchy.

5. In Format 1, data-name-1 or data-name-2 may be a record-name.

6. If explicitly referenced, a paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to from within the same section. A paragraph-name or section-name appearing in a program may not be referenced from any other program, |however a global paragraph-name or section-name may be referenced in GO TO statements from programs contained (directly or not) in the program in which the global paragraph-name or section-name is declared.|

7. If more than one COBOL library is available to the compiler during compilation, text-name must be qualified each time it is referenced; |otherwise standard searching rules apply.|

8. LINAGE-COUNTER must be qualified each time it is referenced if more than one File Description entry containing a LINAGE clause has been specified in the source program.

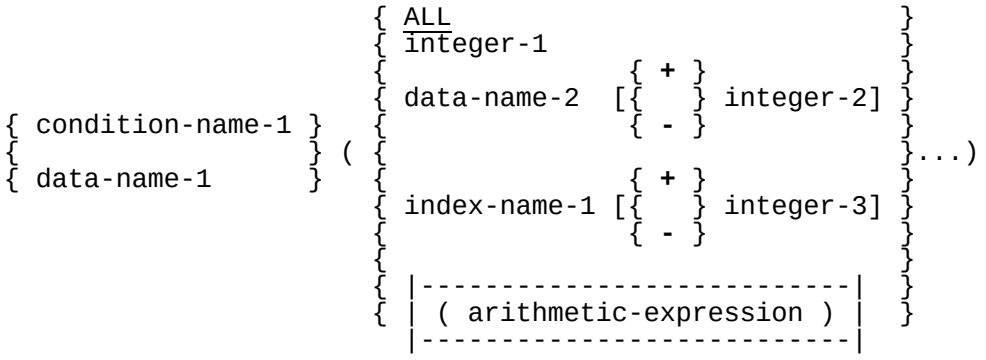
9. PAGE-COUNTER and LINE-COUNTER must be qualified each time they are referenced in the Procedure Division if more than one Report Description entry is specified in the source program. In the Report Section, an unqualified reference to PAGE-COUNTER or LINE-COUNTER is qualified implicitly by the name of the report in whose Report Description entry the reference is made. Whenever the PAGE-COUNTER and LINE-COUNTER of a different report is referenced, PAGE-COUNTER and LINE-COUNTER must be qualified explicitly by the report-name associated with the different report.

3.3.9.2 Subscripting

Function

Subscripts are used when reference is made to an individual element within a table of like elements that have not been assigned individual data-names (see the "OCCURS Clause", Chapter 8).

Format



Syntax Rules

1. The data description entry containing data-name-1 or the data-name associated with condition-name-1 must contain an OCCURS clause or must be subordinate to a data description entry which contains an OCCURS clause.

2. Except as defined in syntax rule 4, when a reference is made to a table element, the number of subscripts must equal the number of OCCURS clauses in the description of the table element being referenced. When more than one subscript is required, the subscripts are written in the order of successively less inclusive dimensions of the table.

3. Index-name-1 must correspond to a data description entry in the hierarchy of the table being referenced which contains an INDEXED BY phrase specifying that index-name.

4. Each table element reference must be subscripted except when such reference appears:

   a. In a USE FOR DEBUGGING statement.

   b. As the subject of a SEARCH statement.

c. In a REDEFINES clause.

d. In a KEY IS phrase of an OCCURS clause.

5. Data-name-2 may be qualified and must be a numeric elementary item representing an integer.

6. Integer-1 may be signed and, if signed, it must be positive.

7. The subscript ALL may be used only when the subscripted identifier is used as a function argument and may not be used when condition-name-1 is specified.

General Rules

1. The value of the subscript must be a positive integer. The lowest possible occurrence number represented by a subscript is 1. The first element of any given dimension of a table is referenced by an occurrence number of 1. Each successive element within that dimension of the table is referenced by occurrence numbers of 2, 3, ... The highest permissible occurrence number for any given dimension of the table is the maximum number of occurrences of the item as specified in the associated OCCURS clause.

2. The value of the index referenced by index-name-1 corresponds to the occurrence number of an element in the associated table.

3. The value of the index referenced by index-name-1 must be initialized before it is used as a subscript. An index may be given an initial value by either a PERFORM statement with the VARYING phrase, a SEARCH statement with the ALL phrase, or a SET statement. An index may be modified only by the PERFORM, SEARCH, and SET statements.

4. If integer-2 or integer-3 is specified, the value of the subscript is determined by incrementing by the value of integer-2 or integer-3 (when the operator + is used) or by decrementing by the value of integer-2 or integer-3 (when the operator - is used) either the occurrence number represented by the value of the index referenced by index-name-1 or the value of the data item referenced by data-name-2.

3.3.9.3 Function-Identifier

Purpose of a Function-Identifier

A function-identifier is a syntactically correct combination of character-strings and separators that uniquely references the data item resulting from the evaluation of a function.

General Format

```
FUNCTION function-name-1 [({argument-1} ... )]
                         [reference-modifier]
```

Syntax Rules

1. Argument-1 must be an identifier, a literal or an arithmetic expression. Specific rules governing the number, class and category of argument-1 are given in the definition of each function.

2. A reference-modifier may be specified only for functions of the category alphanumeric.

3. A function-identifier which references an alphanumeric function may be specified anywhere in the general formats that an identifier is permitted and where the rules associated with the general formats do not specifically prohibit reference to functions, except as follows:

   - As a receiving operand of any statement.

   - Where the rules associated with the general formats require the data item being referenced to have particular characteristics (such as class and category, usage, size, sign and permissible values) and the evaluation of the function according to its definition and the particular arguments specified would not have these characteristics.

4. A function-identifier which references an integer or numeric function may be used only in an arithmetic expression.

General Rules

1. The class and other characteristics of the function being referenced are determined by the function definition.

2. At the time reference is made to a function, its arguments are evaluated individually in the order specified in the list of arguments, from left to right. An argument being evaluated may itself be a function-identifier or may be an expression containing function-identifiers. There is no restriction preventing the function referenced in evaluating an argument from being the same function as that for which the argument is specified.

3.3.9.4 Reference Modifier

Function

Reference modification defines a data item by specifying a leftmost character and length for the data item.

Format

```
{ data-name-1                                      }
{ FUNCTION function-name-1 [({argument-1} ... )] }

                        (leftmost-character-position : [length] )
```

Note: Data-name-1 and FUNCTION function-name-1 (argument-1) are shown in the above format to provide context and are not part of the reference-modifier.

Syntax Rules

1. Data-name-1 must reference a data item whose usage is DISPLAY.

2. Leftmost-character-position and length must be arithmetic expressions.

3. Unless otherwise specified, reference modification is allowed anywhere an identifier referencing a data item of the class alphanumeric |if data-name-1 is of the class alphanumeric, or boolean if data-name-1 is of the class boolean,| is permitted.

4. Data-name-1 may be qualified or subscripted.

5. The function referenced by function-name-1 and its arguments, if any, must be an alphanumeric function.

General Rules

1. Each character of a data item referenced by data-name-1 or by function-name-1 and its arguments, if any, is assigned an ordinal number incrementing by one from the leftmost position to the rightmost position. The leftmost position is assigned the ordinal number one. If the data description entry for data-name-1 contains a SIGN IS SEPARATE clause, the sign position is assigned an ordinal number within that data item.

2. If the data item referenced by data-name-1 is described as |boolean,| numeric, numeric edited, alphanumeric, or alphanumeric edited, it is operated upon for purposes of reference modification as if it were redefined as an alphanumeric data item of the same size as the data item referenced by data-name-1.

3. Reference modification for an operand is evaluated immediately after evaluation of any subscripts that are specified for that operand.

   If an ALL subscript is specified for an operand, the reference-modifier is applied to each of the implicitly specified elements of the table.

   If reference modification is specified in a function reference, the reference modification is evaluated immediately after evaluation of the function.

4. Reference modification creates a unique data item which is a subset of the data item referenced by data-name-1 or by function-name-1 and its arguments, if any. This unique data item is defined as follows:

   a. The evaluation of leftmost-character-position specifies the ordinal position of the leftmost character of the unique data item in relation to the leftmost character of the data item referenced by data-name-1 or function-name-1 and its arguments, if any. Evaluation of leftmost-character-position must result in a positive non-zero integer less than or equal to the number of characters in the data item referenced by data-name-1 or function-name-1 and its arguments, if any.

   b. The evaluation of length specifies the size of the data item to be used in the operation. The evaluation of length must result in a positive non-zero integer. The sum of leftmost-character-position and length minus the value one must be less than or equal to the number of characters in the data item referenced by data-name-1 or function-name-1 and its arguments, if any. If length is not specified, the unique data item extends from and includes the character identified by leftmost-character-position up to and including the rightmost character of the data item referenced by data-name-1 or function-name-1 and its arguments, if any.

5. The unique data item is considered an elementary data item without the JUSTIFIED clause. When a function is referenced, the unique data item has the class and category of alphanumeric. When data-name-1 is specified, the unique data item has the same class and category as that defined for the data item referenced by data-name-1 except that the categories numeric, numeric edited, and alphanumeric edited are considered category alphanumeric.

3.3.9.5 Identifier

An identifier is a syntactically correct sequence of character-strings and separators used to reference data uniquely.

When a data item other than a function is being referenced, identifier is a term used to reflect that data-name, if not unique in a program, must be followed by a syntactically correct combination of qualifiers, subscripts, or reference modifiers necessary for uniqueness of reference.

General Format

Format 1

```
function-identifier-1
```

Format-2

```
                                             {cd-name     }
               {IN}                    {IN}  {            }
data-name-1 [  {  }  data-name-2]... [{  }  {file-name   }]
               {OF}                    {OF}  {            }
                                             {report-name}

            [ ( {subscript}... )]

            [ (leftmost-character-position : [length] )]
```

Syntax Rule

The words IN and OF are equivalent.

3.3.9.6 Condition-Name

If explicitly referenced, a condition-name must be unique or be made unique through qualification and/or subscripting except when the scope of names conventions by themselves ensure uniqueness of reference (See "Scope of Names", this chapter).

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

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

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

In the general formats of the chapters that follow, "condition-name" refers to a condition-name qualified or subscripted, as necessary.

3.4 EXPLICIT AND IMPLICIT SPECIFICATIONS

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

1. Explicit and implicit Procedure Division references
2. Explicit and implicit transfers of control
3. Explicit and implicit attributes
4. Explicit and implicit scope terminators.

3.4.1 Explicit and Implicit Procedure Division References

A COBOL source program can reference data items either explicitly or implicitly in Procedure Division statements. An explicit reference occurs when the name of the referenced item is written in a Procedure Division statement or when the name of the referenced item is copied into the Procedure Division by the processing of a COPY statement. An implicit reference occurs when the item is referenced by a Procedure Division statement without the name of the referenced item being written in the source statement. An implicit reference also occurs, during the execution of a PERFORM statement, when the index or data item referenced by the index-name or identifier specified in the VARYING, AFTER or UNTIL phrase is initialized, modified or evaluated by the control mechanism associated with that PERFORM statement. Such an implicit reference occurs if and only if the data item contributes to the execution of the statement.

3.4.2 Explicit and Implicit Transfers of Control

The mechanism that controls program flow transfers control from statement to statement in the sequence in which they were written in the source program unless an explicit transfer of control overrides this sequence or there is no next executable statement to which control can be passed. The transfer of control from statement to statement occurs without the writing of an explicit Procedure Division statement, and, therefore, is an implicit transfer of control.

COBOL provides both explicit and implicit means of altering the implicit control transfer mechanism.

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

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

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

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

An explicit transfer of control consists of an alteration of the implicit control transfer mechanism by the execution of a procedure branching or conditional statement (See "Categories of Statements", Chapter 10). An explicit transfer of control can be caused only by the execution of a procedure branching or conditional statement. The procedure branching statement EXIT PROGRAM causes an explicit transfer of control only when the statement is executed in a called program.

In this manual, the term "next executable statement" is used to refer to the next COBOL statement to which control is transferred according to the rules above and the rules associated with each language element.

There is no next executable statement when the program contains no Procedure Division or following:

1. The last statement in a declarative section when the paragraph in which it appears is not being executed under the control of some other COBOL statement.

2. The last statement in a declarative section when the statement is in the range of an active perform statement executed in a different section and this last statement of the declarative section is not also the last statement of the procedure that is the exit of the active PERFORM statement.

3. The last statement in a program when the paragraph in which it appears is not being executed under the control of some other COBOL statement in that program.

4. A STOP RUN statement or EXIT PROGRAM statement that transfers control outside the COBOL program.

5. The end program header.

When there is no next executable statement and control is not transferred outside the COBOL program, the program flow of control is undefined unless the program execution is in the non-declarative procedures portion of a program under control of a CALL statement, in which case an implicit EXIT PROGRAM statement is executed.

3.4.3 Explicit and Implicit Attributes

Attributes may be implicitly or explicitly specified. Any attribute which has been explicitly specified is called an explicit attribute. If an attribute has not been specified explicitly, then the attribute takes on the default specification. Such an attribute is known as an implicit attribute.

For example, the usage of a data item need not be specified, in which case a data item's usage is DISPLAY.

3.4.4 Explicit and Implicit Scope Terminators

Scope terminators serve to delimit the scope of certain Procedure Division statements (See "Delimited Scope Statements", Chapter 10).

Scope terminators are of two types: explicit and implicit.

The explicit scope terminators are the following:

END-ADD	END-MULTIPLY	END-SEARCH
END-CALL	END-PERFORM	END-START
END-COMPUTE	END-READ	END-STRING
END-DELETE	END-RECEIVE	END-SUBTRACT
END-DIVIDE	END-RETURN	END-UNSTRING
END-EVALUATE	END-REWRITE	END-WRITE
END-IF		

The implicit scope terminators are the following:

1. At the end of any sentence, the separator period which terminates the scope of all previous statements not yet terminated.

2. Within any statement containing another statement, the next phrase of the containing statement following the contained statement terminates the scope of any unterminated contained statement. Examples of such phrases are ELSE, WHEN, NOT AT END, etc.

3.5 ACCESSING DATA ITEMS

Data items may be classified according to which programs in a run unit may access them.

The following classifications, which are disjoint, arise:

1. External data records and items.

2. Local data items.

In addition, if a data item can be referenced by a program, it may be passed as a parameter to other programs called by that program.

3.5.1 External Data Records and Items

An external data record is a logical record which is described in the Working-Storage or the Constant Section of one or more programs of a run unit. An external data item is a data item which constitutes a part of an external data record. External data items and external data records may be referenced from any program which describes them. An external data record attains the external attribute by including the EXTERNAL clause in the Data Description entry for that record in each program which describes that record. An external data item attains the external attribute by including its Data Description entry within the description of an external record.

3.5.2 Local Data Items

A local data item is a data item which is described without the EXTERNAL clause in a program.

3.6 EXTERNAL SWITCH

An external switch is a JCL switch which is used to indicate that one of two alternate states exist. These alternate states are referred to as the "on" status and the "off" status of the associated external switch. The status of an external switch may be interrogated by testing condition-names associated with that switch. The association of a condition-name with an external switch and the association of a user-specified mnemonic-name with SWITCH-n that names an external switch is established in the SPECIAL-NAMES paragraph of the Environment Division (see Chapter 7). The scope of an external switch is the run unit. SWITCH-n that names such an external switch refers to one and only one such switch, the status of which is available to each object program functioning within that run unit. The status of certain external switches may be altered by the SET statement (See the "SET Statement", Chapter 13).

3.7 SCOPE OF NAMES

When programs are directly or indirectly contained within other programs, each program may use identical user-defined words to name objects independent of the use of these user-defined words by other programs (See "User-Defined Words", this chapter). When identically named objects exist, a program's reference to such a name, even when it is a different type of user-defined word, is to the object which that program describes rather than to the object, possessing the same name, described in another program.

The following types of user-defined words may be referenced only by statements and entries in that program in which the user-defined word is declared |except when the name is global and the program contains other programs:|

1. cd-name
2. paragraph-name
3. section-name

The following types of user-defined words may be referenced by any COBOL program:

1. library-name
2. text-name

The following types of user-defined words when they are declared in a Communication Section may be referenced only by statements and entries in that program which contains that section |except when the name is global and the program contains other programs:|

1. condition-name
2. data-name
3. record-name

The following types of names, when they are declared within a Configuration Section, may be referenced only by statements and entries either in that program which contains a Configuration Section or in any program contained within that program:

1. alphabet-name
2. class-name
3. condition-name
4. mnemonic-name
5. symbolic-character

Specific conventions, for declarations and references, apply to the following types of user-defined words when the conditions listed above do not apply:

1. condition-name
2. data-name
3. file-name
4. index-name
5. program-name
6. record-name
7. report-name
8. cd-name
9. paragraph-name
10. section-name

3.7.1 Conventions for Program-names

The program-name of a program is declared in the PROGRAM-ID paragraph of the program's Identification Division. A program-name may be referenced only by the CALL statement, the CANCEL statement, and the end program header. The program-names allocated to programs constituting a run unit are not necessarily unique but, when two programs in a run unit are identically named, at least one of those two programs must be directly or indirectly contained within another separately compiled program which does not contain the other of those two programs.

The following rules regulate the scope of a program-name.

1. If the program-name is that of a program which does not possess the common attribute and which is directly contained within another program, that program-name may be referenced only by statements included in that containing program.

2. If the program-name is that of a program which does possess the common attribute and which is directly contained within another program, that program-name may be referenced only by statements included in that containing program and any programs directly or indirectly contained within that containing program, except that program possessing the common attribute and any programs contained within it.

3. If the program-name is that of a program which is separately compiled, that program-name may be referenced by statements included in any other program in the run unit, except programs it directly or indirectly contains.

3.7.2 Conventions for Index-names

If a data item possessing either or both the external or global attributes includes a table accessed with an index, that index also possesses correspondingly either or both attributes. Therefore, the scope of an index-name is identical to that of the data-name which names the table whose index is named by that index-name and the scope of name rules for data-names apply. Index-names cannot be qualified.

3.7.3 Conventions for Other Names

The conventions for Conditions-Names, Data-names, File-names, Record-names, Report-Names, |Cd-names, Paragraph-names, and Section-Names| are given below.

When condition-names, data-names, file-names, record-names, report-names, |cd-names, paragraph-names, and section-names| are declared in a source program, these names may be referenced only by that program except when one or more of the names is global and the program contains other programs.

The requirements governing the uniqueness of the names allocated by a single program to be condition-names, data-names, file-names, record-names, report-names, |cd-names, paragraph-names, and section-names| are explained elsewhere in these specifications (See "User-Defined Words", this chapter).

A program cannot reference any condition-name, data-name, file-name, record-name, report-name, |cd-name paragraph-name, or section-name| declared in any program it contains.

A global name may be referenced in the program in which it is declared or in any programs which are directly or indirectly contained within that program.

When a program, program B, is directly contained within another program, program A, both programs may define a condition-name, a data-name, a file-name, a record-name, a report-name, a |cd-name, a paragraph-name, or a section-name| using the same user-defined word. When such a duplicated name is referenced in program B, the following rules are used to determine the referenced object:

1. The set of names to be used for determination of a referenced object consists of all names which are defined in program B and all global names which are defined in program A and in any programs which directly or indirectly contain program A. Using this set of names, the normal rules for qualification and any other rules for uniqueness of reference are applied until one or more objects is identified.

2. If only one object is identified, it is the referenced object.

3. If more than one object is identified, no more than one of them can have a name local to program B. If zero or one of the objects has a name local to program B, the following rules apply:

    a. If the name is declared in program B, the object in program B is the referenced object.

    b. Otherwise, if program A is contained within another program, the referenced object is:

        1) The object in program A if the name is declared in program A.

        2) The object in the containing program if the name is not declared in program A and is declared in the program containing program A. This rule is applied to further containing programs until a single valid name has been found.

|If the referenced object is a global paragraph-name or a global section-name and it is not declared in the program which contains the reference, this reference is legal only if it appears in a GO TO statement excluding any DEPENDING phrase.|

4. The COBOL Program: a Summary

A COBOL source program is a syntactically correct set of COBOL statements.

With the exception of COPY and REPLACE statements and the end program header, the statements, entries, paragraphs, and sections of a COBOL source program are grouped into the following divisions: |Control,| Identification, Environment, Data, and Procedure.

The end of a COBOL program is indicated either by the end program header, if specified, or by the absence of additional source program lines.

The purpose and general composition of the COBOL divisions are summarized below. The end program header structure is also presented. Specific language and usage requirements for each COBOL statement and term appear in Chapters 5 through 13.

4.1 STRUCTURE OF A COBOL PROGRAM

The following gives the general format and order of presentation of the entries and statements which constitute a COBOL source program.

4.1.1 General Format

```
[control-division]

identification-division

[environment-division]

[data-division]

[procedure-division]

[end-program-header]
```

4.1.2 Syntax Rule

The generic terms |control-division,| identification-division, data-division, procedure-division, and end-program-header represent |a COBOL Control Division,| a COBOL Identification Division, a COBOL Environment Division, a COBOL Data Division, a COBOL Procedure Division, and a COBOL end program header, respectively.

4.1.3 General Rules

1. The beginning of a division in a program is indicated by the appropriate division header. The end of a division is indicated by one of the following:

   a. The division header of a succeeding division in that program.

   b. The end program header.

   c. That physical position after which no more source program lines occur.

2. All separately compiled source programs in a sequence of programs must be terminated by an end program header except for the last program of the sequence.

4.2 CONTROL DIVISION

|The Control Division consists of the Substitution Section and the Default Section.

The Control Division directs the compiler to replace specified words or literals in the source program with other words at program compilation using the Substitution Section. The Default Section allows the compiler default conditions to be specified if other than standard defaults are required.|

4.3 IDENTIFICATION DIVISION

The Identification Division uniquely identifies a COBOL program. It contains the program-name, which names the source program input, the listing of the compiled program that is printed out and the object compile unit. The Identification Division can indicate the author of the program, the installation where the program is compiled, the date on which the program is written and the date on which it is compiled. It can also include security information.

4.4 ENVIRONMENT DIVISION

The Environment Division specifies a standard method of expressing those aspects of a data processing problem that are dependant 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.

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

The Configuration Section describes the computer configuration on which the source program is to be compiled, and the configuration on which the compiled program is to be run. It also relates system names used by the compiler to names introduced by the programmer in the source program.

The Input-Output Section contains the information needed to control transmission and handling of data between external media and the object program. This section describes the name, type of organization, and access mode of each data file, and may associate the file with a peripheral device. It may also designate memory areas to be shared by files.

4.5 DATA DIVISION

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

1. Data that is contained in files and enters a specified area of the internal memory of the computer or leaves memory from a specified area.

2. Data that is developed internally and placed into intermediate or working storage, or placed into specific format for output reporting purposes.

3. Data whose value is assigned in the source program.

4. Data made available to one program through another program containing a CALL statement.

The sections of the Data Division are the File, Working-Storage, |Constant,| Linkage, Communication and Report Sections.

The File Section describes the structure of data files; each file is defined by a file description entry and one or more record description entries.

The Working-Storage Section describes records and non-contiguous data items that are not part of external files, but are developed and processed internally.

|The Constant Section defines data items whose values do not change during the execution of the program.|

The Linkage Section of a COBOL program is meaningful if the program is to be called by another program in the same run unit. This section, appearing in the called program, describes data items that may be referred to by both the called and calling programs, but are available only through the calling program. If the program uses based data items, descriptions of these items are to be found in the Linkage Section.

The Communication Section describes the characteristics of the information that is exchanged between the program and the communication system during communication processing.

The Report Section describes the content and format of Reports that are to be generated.

4.6 PROCEDURE DIVISION

The Procedure Division contains the specific instructions for solving a data processing problem, using the data described in the Data Division. The Procedure Division is written as a group of consecutive procedures, each composed of a series of closely related operations that collectively perform a particular function.

This division comprises two kinds of procedures: declarative procedures, which are optional, and non-declarative procedures. If used, declarative procedures must all be grouped at the beginning of the Procedure Division. Use of the declarative procedures permits the execution of procedures that are not performed in the regular sequence of coding, but are initiated as a result of a condition that the programmer cannot test directly. Each declarative procedure must contain a compiler-directing sentence and may include one or more paragraphs.

A procedure is composed of a paragraph, or a group of successive paragraphs, 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 section consists of a section header followed by zero, one or more successive paragraphs.

A paragraph consists of a paragraph-name followed by a period followed by a space and by zero, one, or more successive sentences.

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.

Execution of the procedures in the Procedure Division begins with the first statement in the 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.

4.7 END PROGRAM HEADER

The end program header indicates the end of the named COBOL program.

4.7.1 Format

```
END PROGRAM  program-name.
```

4.7.2 Syntax Rules

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

2. The program-name must be identical to a program-name declared in a preceding PROGRAM-ID paragraph.

4.7.3 General Rules

1. The end program header indicates the end of the specified COBOL source program.

2. If the next source statement after the program terminated by the end program header is a COBOL statement, |it can be any COBOL statement pertaining to the Procedure Division of a containing program or| it must be |the Control Division or| the Identification Division header of a program to be compiled separately from that program terminated by the end program header.

5. Control Division

5.1 GENERAL DESCRIPTION

The Control Division provides for the substitution, replacement and modification of source program text; and declaration of default options for the computer system for which the program is to be compiled.

5.2 CONTROL DIVISION

Description

The Control Division directs the compiler to replace COBOL text within the source program at program compilation using the Substitution Section. The Default Section allows the compiler default conditions to be specified if other than standard defaults are required.

Format

```
|-------------------------------------------|
| CONTROL DIVISION.                         |
|                                           |
| [SUBSTITUTION SECTION. [replace-entry]]   |
|                                           |
| [DEFAULT SECTION. [[default-entry]... . ]] |
|-------------------------------------------|
```

Syntax Rules

1. The Control Division is optional. If it is specified, it must appear before the Identification Division.

2. The Control Division must not be stated in a program which is contained directly or indirectly within another program.

   The entries stated in the Control Division of a program which contains other programs apply to each contained program.

3. The Control Division must not contain nor be preceded by any REPLACE statement. Further, if the replace-entry is present in the Substitution Section of the Control Division of a source program, that source program, including all contained programs, must contain no REPLACE statement. (See the "REPLACE Statement", Chapter 15).

4. COPY statements must not appear in the replace-entry. However, a replace-entry may be part of the text copied by a COPY statement provided that this COPY statement does not have the REPLACING option.

General Rules

1. All COPY statements appearing in the source program are resolved before any CONTROL DIVISION entries.

2. Resolution of the entries appearing in the Control Division is accomplished in the following order:

   a. The entry in the Substitution Section
   b. All entries in the Default Section.

5.3 SUBSTITUTION SECTION

Description

The Substitution Section provides the means of replacing source program text by other source program text.

Format

```
|-------------------------------------------------------------|
|                                                             |
| SUBSTITUTION SECTION                                        |
|    [ REPLACE                                                |
|          {== pseudo-text-1 ==}    {== pseudo-text-2 ==}     |
|          {identifier-1       }    {identifier-2       }     |
|         {{                   } BY {                   }}    |
|         {{literal-1          }    {literal-2          }}    |
|         {{word-1             }    {word-2             }}... .]|
|         {                                              }    |
|         {{LEADING }              {literal}             }    |
|         {{        } literal-3 BY {SPACE  }             }    |
|          {TRAILING}              {SPACES }                  |
|                                                             |
|-------------------------------------------------------------|
```

Syntax Rules

1. Pseudo-text-1 must contain one or more text-words.

2. Pseudo-text-2 may contain zero, one or more text-words.

3. Character-strings within pseudo-text-1 and pseudo-text-2 may be continued. However, both characters of a pseudo-text delimiter must be on the same line.

4. Word-1 or word-2 may be any single COBOL word, except PICTURE and PIC.

5. Literal-3 and literal-4 must be non-numeric literals.

6. A PICTURE character-string may appear in pseudo-text-1 or pseudo-text-2 provided it is immediately preceded by "PICTURE", "PICTURE IS", "PIC", or "PIC IS".

7. The words PICTURE or PIC may appear in pseudo-text-1 or pseudo-text-2 provided they are immediately followed by a PICTURE character-string possibly preceded by the word IS.

8. If the Substitution Section contains one or more REPLACE clause(s), the source program must contain no REPLACE statement. (See the "REPLACE Statement", Chapter 15).

General Rules

1. The compilation of a source program containing a REPLACE clause in the Substitution Section is logically equivalent to processing all COPY statements to produce an intermediate source program, and then processing all REPLACE operands against the intermediate source program to produce the final source program.

2. When a REPLACE clause is processed, the (possibly intermediate) source program text is searched and each properly matched occurrence of pseudo-text-1, identifier-1, literal-1, word-1 and leading or trailing literal-3 in words in the source program text is replaced by the corresponding pseudo-text-2, identifier-2, literal-2, word-2, literal-4, or deleted.

3. For purposes of matching, identifier-1, literal-1, and word-1 are treated as pseudo-text containing only identifier-1, literal-1, or word-1, respectively.

4. The comparison operation to determine text replacement occurs in the following manner:

   a. Starting with the leftmost source program text-word and the first pseudo-text-1, identifier-1, literal-1, word-1, or literal-3 that was specified in the REPLACE clause, the entire REPLACE clause operand that precedes the reserved word BY is compared to an equivalent number of contiguous source program text-words, or in the case of literal-3 to the equivalent number of LEADING or TRAILING characters of the leftmost text-word when it is a word.

   b. Pseudo-text-1, identifier-1, literal-1, or word-1 match the source program text if, and only if, the ordered sequence of text-words that forms pseudo-text-1, identifier-1, literal-1, or word-1 is equal, character for character, to the ordered sequence of source program text-words. For purposes of matching, each occurrence of a separator comma or semi-colon in pseudo-text-1 or in the source program text is considered to be a single space except when pseudo-text-1 consists solely of either a separator comma or semi-colon, in which case it participates in the match as a text-word. Each sequence of one or more space separators is considered to be a single space.

   c. Literal-3 matches the source program text if, and only if, the leftmost text-word commences (LEADING) or finishes (TRAILING) with the same sequence of characters as that forms literal-3.

   d. If no match occurs, the comparison is repeated with each next successive pseudo-text-1, identifier-1, word-1, literal-1, or literal-3, if any, in the REPLACE clause until either a match is found or there is no next successive REPLACE operand.

   e. When all the REPLACE clause operands have been compared and no match has occurred, the leftmost source program text-word remains unchanged. The next successive source program text-word is then considered as the leftmost source program text-word, and the comparison cycle starts again with the first pseudo-text-1, identifier-1, word-1, literal-1, or literal-3 specified in the REPLACE clause.

f. Whenever a match occurs between pseudo-text-1, identifier-1, word-1 or literal-1, and the source program text, the corresponding pseudo-text-2, identifier-2, word-2 or literal-2 is placed into the source program. Whenever a match occurs between literal-3 and a text-word, the matching leading or trailing characters of the word are either replaced by the characters that form literal-4, or if the SPACE or SPACES phrase is used, are deleted; the replacement is effected only if it results in a legal word. The source program text-word immediately following the rightmost text-word that participated in the match is then considered as the leftmost source program text-word. The comparison cycle starts again with the first pseudo-text-1, identifier-1, word-1, literal-1 or literal-3 specified in the REPLACE clause.

g. The comparison operation continues until the rightmost text-word in the source program text has either participated in a match or been considered as a leftmost source program text-word and participated in a complete comparison cycle.

5. A comment line occurring in the source program text or pseudo-text-1 is interpreted, for purposes of matching, as a single space. Comment lines appearing in pseudo-text-2 and library text are copied into the source program unchanged.

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

7. The syntactic correctness of the entire COBOL source program cannot be determined until all REPLACE clauses have been completely processed.

8. Each word in pseudo-text-2 that is to be placed in the resultant program is placed in the same area of the resultant program as it appears in pseudo-text-2. Literal-2, word-2 or the first word of identifier-2, is placed in the same area as the first text-word replaced.

9. For purposes of compilation, text-words after replacement are placed in the source program according to the rules for the reference format.

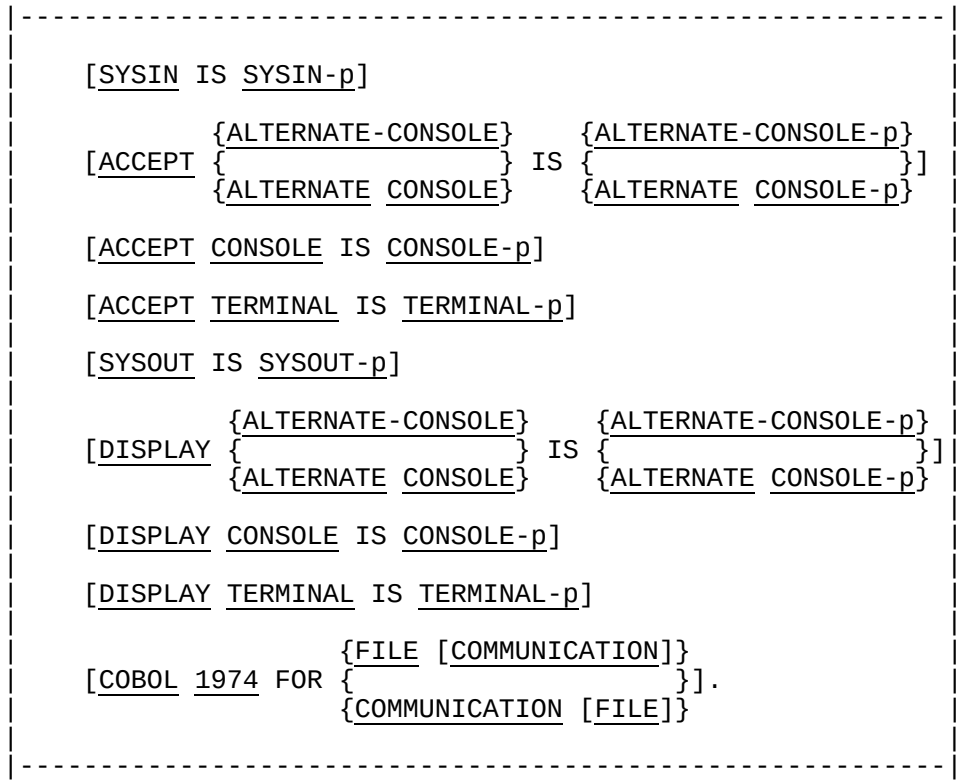
5.4 DEFAULT SECTION

DESCRIPTION

The Default Section specifies language elements or functions that are to be used as defaults during compilation of the program.

Format

```
|-------------------------------------------------------|
|                                                       |
|DEFAULT SECTION.                                       |
|                          {OMITTED     }               |
|    [SYMBOLIC QUEUE IS {             }]               |
|                          {MESSAGE UNIT}               |
|                                                       |
|                        {LEADING }                     |
|    [DISPLAY SIGN IS {         } [SEPARATE CHARACTER]] |
|                        {TRAILING}                     |
|                                                       |
|                          {BINARY         }            |
|                          {DISPLAY        }            |
|                          {COMPUTATIONAL-1}            |
|                          {COMP-1         }            |
|                          {COMPUTATIONAL-2}            |
|    {COMPUTATIONAL}       {COMP-2         }            |
|   [{              } IS {COMPUTATIONAL-3}]            |
|    {COMP          }      {COMP-3         }            |
|                          {COMPUTATIONAL-5}            |
|                          {COMP-5         }            |
|                          {COMPUTATIONAL-8}            |
|                          {COMP-8         }            |
|                          {PACKED-DECIMAL }            |
|                                                       |
|                          {NOT STANDARD}               |
|              {integer [{             }]}              |
|              {         {BINARY      } }              |
|    [TEMP IS {                         }]             |
|              {NOT STANDARD            }              |
|              {BINARY                  }              |
|                                                       |
|                { SYSIN              }                 |
|                {[ALTERNATE] CONSOLE}                  |
|    [ACCEPT IS {                     }]                |
|                { ALTERNATE-CONSOLE  }                 |
|                { TERMINAL           }                 |
|                 { SYSOUT            }                 |
|                 {[ALTERNATE] CONSOLE}                 |
|    [DISPLAY IS {                    }]                |
|                { ALTERNATE-CONSOLE  }                 |
|                { TERMINAL           }                 |
|                                                       |
|-------------------------------------------------------|
```



Syntax Rules

1. Integer must range from 1 through 30.

2. COMP is an abbreviation for COMPUTATIONAL.

   COMP-n is an abbreviation for COMPUTATIONAL-n.

3. The suffix -p in ALTERNATE-CONSOLE-p, CONSOLE-p, SYSIN-p, SYSOUT-p and TERMINAL-p may be -0, -1, -2 or -X.

General Rules

1. The SYMBOLIC QUEUE clause is given for documentation only. It is accepted for compatibility.

2. The DISPLAY SIGN clause specifies for DISPLAY numeric data the default position and mode of representation of the operational sign. If this clause is not present, DISPLAY SIGN IS TRAILING is assumed.

3. The COMPUTATIONAL clause specifies the format of a data item described with the USAGE IS COMPUTATIONAL clause, in the computer storage. If this clause is not present COMPUTATIONAL IS COMPUTATIONAL-3 is assumed.

4. The TEMP clause specifies the number of significant digit positions kept in intermediate results of arithmetic expressions. If this clause is not present, or if the integer phrase is not present in the clause, TEMP IS 30 is assumed, unless the compilation level is ANSI or below in which case TEMP IS 18 is assumed. When NOT STANDARD is specified, the number of significant digits is not always guaranteed when faster computations may be achieved. When BINARY is specified, intermediate computations will use binary floating point intermediate results.

5. The ACCEPT clause specifies the standard device from which data is made available when the ACCEPT statement is used without the FROM phrase. When this clause is absent, ACCEPT IS SYSIN is assumed.

6. The DISPLAY clause specifies the standard device to which data is transferred when the DISPLAY statement is used without the UPON phrase. When this clause is absent, DISPLAY IS SYSOUT is assumed.

7. The SYSIN clause specifies the legible equivalent for ACCEPT statements that reference either SYSIN or a mnemonic-name associated with SYSIN. If this clause is not present, SYSIN IS SYSIN-1 is assumed. (See "Legible Equivalent", Chapter 3).

8. The ACCEPT ALTERNATE-CONSOLE or ACCEPT ALTERNATE CONSOLE clause specifies the legible equivalent for ACCEPT statements that reference either ALTERNATE-CONSOLE or ALTERNATE CONSOLE or a mnemonic-name associated with ALTERNATE-CONSOLE or ALTERNATE CONSOLE. If this clause is not present, ACCEPT ALTERNATE-CONSOLE IS ALTERNATE-CONSOLE-2 is assumed. (See "Legible Equivalent", Chapter 3).

9. The ACCEPT CONSOLE clause specifies the legible equivalent for ACCEPT statements that reference either CONSOLE or a mnemonic-name associated with CONSOLE. If this clause is not present, ACCEPT CONSOLE IS CONSOLE-2 is assumed. (See "Legible Equivalent", Chapter 3).

10. The ACCEPT TERMINAL clause specifies the legible equivalent for ACCEPT statements that reference either TERMINAL or a mnemonic-name associated with TERMINAL. If this clause is not present, ACCEPT TERMINAL IS TERMINAL-2 is assumed. (See "Legible Equivalent", Chapter 3).

11. The SYSOUT clause specifies the legible equivalent for DISPLAY statements that reference either SYSOUT or a mnemonic-name associated with SYSOUT. If this clause is not present, SYSOUT IS SYSOUT-1 is assumed. (See "Legible Equivalent", Chapter 3).

12. The DISPLAY ALTERNATE-CONSOLE or DISPLAY ALTERNATE CONSOLE clause specifies the legible equivalent for DISPLAY statements that reference either ALTERNATE-CONSOLE or ALTERNATE CONSOLE or a mnemonic-name associated with ALTERNATE-CONSOLE or ALTERNATE CONSOLE. If this clause is not present, DISPLAY ALTERNATE-CONSOLE IS ALTERNATE-CONSOLE-2 is assumed. (See "Legible Equivalent", Chapter 3).

13. The DISPLAY CONSOLE clause specifies the legible equivalent for DISPLAY statements that reference either CONSOLE or a mnemonic-name associated with CONSOLE. If this clause is not present, DISPLAY CONSOLE IS CONSOLE-2 is assumed. (See "Legible Equivalent", Chapter 3).

14. The DISPLAY TERMINAL clause specifies the legible equivalent for DISPLAY statements that reference either TERMINAL or a mnemonic-name associated with TERMINAL. If this clause is not present, DISPLAY TERMINAL IS TERMINAL-2 is assumed. (See "Legible Equivalent", Chapter 3).

15. The COBOL 1974 clause specifies that the syntax and semantic rules of COBOL 74 apply for the entries, clauses and statements related to files (if FILE is specified) and communications (if COMMUNICATION is specified). However, the COBOL 1974 clause does not prevent from using the GLOBAL clause in the File and Communication Sections, the GLOBAL phrase in USE statements and the scope terminators and the NOT phrases in input/output statements. The Report Writer is not affected by the presence of any COBOL 1974 clause.

6. Identification Division

6.1 GENERAL DESCRIPTION

The Identification Division must be included in every source program. This division identifies the program. 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.

6.2 IDENTIFICATION DIVISION

Description

The Identification Division identifies the source program and the resultant output listing, and may document other related information, as desired.

The AUTHOR paragraph, INSTALLATION paragraph, DATE-WRITTEN paragraph, DATE-COMPILED paragraph and SECURITY paragraph are obsolete elements in Standard COBOL because they are to be deleted from the next revision of Standard COBOL.

Organization

Fixed paragraph names identify the type of information contained in the paragraph. The PROGRAM-ID and DATE-COMPILED paragraphs are defined separately on the following pages. Although the other paragraphs are not declared, their format is similar.

Format

The Identification Division must conform to the following format:

```
IDENTIFICATION DIVISION

                                {COMMON [INITIAL]}
PROGRAM-ID. program-name [IS {                  } PROGRAM].
                                {INITIAL [COMMON]}

[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. The continuation of the comment-entry by the use of the hyphen in the indicator area is not permitted; however, the comment-entry may be contained on one or more lines.

6.3 PROGRAM-ID

Description

The PROGRAM-ID paragraph gives the name by which the program is identified and assigns selected program attributes to that program.

Format

```
                                 {COMMON [INITIAL]}
PROGRAM-ID. program-name [IS {                    } PROGRAM].
                                 {INITIAL [COMMON]}
```

Syntax Rules

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

2. A unique program-name must be assigned to every program contained directly or indirectly within the same separately compiled program.

3. The COMMON phrase may be used only if the program is contained within another program.

General Rules

1. The program-name identifies the source program, the object program, and all listings pertaining to a particular program.

2. The COMMON phrase specifies that the program has the common attribute. A common program is contained within another program but may be called from programs other than that containing it. (See "Scope of Names", Chapter 3.)

3. The INITIAL phrase specifies that the program has the initial attribute. When an initial program is called, it and any program contained within it are placed in their initial state. (See "Initial State of a Program", in the Glossary).

6.4 DATE-COMPILED

Description

The DATE-COMPILED paragraph provides the compilation date in the Identification Division source program listing. The DATE-COMPILED paragraph is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

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 the use of the hyphen in the indicator area is not permitted; however, the comment entry may be contained on one or more lines.

General Rule

The paragraph-name DATE-COMPILED causes the current date to be inserted during program compilation. If a DATE-COMPILED paragraph is present, it is replaced during compilation with a paragraph of the form:

```
DATE-COMPILED. current-date.
```

7. Environment Division

7.1 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 specified 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 may be specified.

The Environment Division is optional in a COBOL source program.

7.2 ORGANIZATION

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

The Configuration Section specifies 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 executed; the SPECIAL-NAMES paragraph, which provides a means for specifying the currency sign, choosing the decimal point, specifying symbolic characters, relating implementor-names to user-specified mnemonic names, relating alphabet-names to character sets or collating sequences, and relating class-names to sets of characters.

The Input-Output Section specifies 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.

7.3 ENVIRONMENT DIVISION

Description

The Environment Division defines those aspects of a program that are dependent upon hardware configurations and considerations.

Format

The following is the general format of the sections and paragraphs in the Environment Division, and defines the order of presentation in the source program. The definitions of the entries for the contents of the paragraphs shown below are given on the following pages.

```
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].]]]
```

Syntax Rule

The Configuration Section must not be stated in a program which is contained directly or indirectly within another program.

General Rule

The entries explicitly or implicitly stated in the Configuration Section of a program which contains other programs apply to each contained program.

7.4 SOURCE-COMPUTER

Description

The SOURCE-COMPUTER paragraph provides a means of describing the computer upon which the program is to be compiled.

Format

```
 SOURCE-COMPUTER.
 {    DPS7                          }
 { |-----------------|               }
 { |                 | {GCOS    } }
[{ | [HIS-SERIES-60] | {        } }
 { |                 | {LEVEL-64} }
 { |-----------------|               }
 {                                  }
 {    computer-name                 }

   |-------------------------------------------------------------|
   |                        {WORDS     }                         |
   |                        {CHARACTERS}                         |
   |              {integer {          }                    }     |
   |              {        {MODULES   }                    }     |
   |              {        {BYTES     }                    }     |
   | [MEMORY SIZE {                                      }]    |
   |              {ADDRESS                               }     |
   |              {               {THROUGH}              }     |
   |              {    {literal-1 {       } literal-2}... }     |
   |                              {THRU   }                      |
   |-------------------------------------------------------------|

     [WITH DEBUGGING MODE]. ]
```

Syntax Rules

1. Computer-name may be any user-defined word.

2. Each literal may be numeric or non-numeric; when numeric, it must be an unsigned integer. Figurative constants are not allowed.

3. The words THROUGH and THRU are equivalent.

General Rules

1. The MEMORY SIZE clause is given for documentation purposes only. It is accepted for compatibility.

2. All clauses of the SOURCE-COMPUTER paragraph apply to the program in which they are explicitly or implicitly specified and to any program contained within that program.

3. When the SOURCE-COMPUTER paragraph is not specified and the program is not contained within a program including a SOURCE-COMPUTER paragraph, the source computer is DPS7.

4. When the SOURCE-COMPUTER paragraph is specified, but the source-computer-entry is not specified, the effect is as though the SOURCE-COMPUTER paragraph is replaced in the source program with a paragraph of the form:

   SOURCE-COMPUTER. DPS7.

5. The WITH DEBUGGING MODE clause serves as a compile-time switch over the debugging statements of the program (See "The Debugging Facility", Chapter 16).

   a. When the WITH DEBUGGING MODE clause is specified in a program and the NDEBUGMD option is not used in the CBL JCL statement, any USE FOR DEBUGGING statement, all associated debugging sections, and all statements contained in all debugging lines in that program and in any program contained in that program are compiled as part of the object program.

   b. When the WITH DEBUGGING MODE clause is not specified in a program nor in any program that contains it, directly or indirectly, and the DBUGMD option is not used in the CBL JCL statement, any USE FOR DEBUGGING statement, all associated debugging sections, and all debugging lines are considered as comment lines.

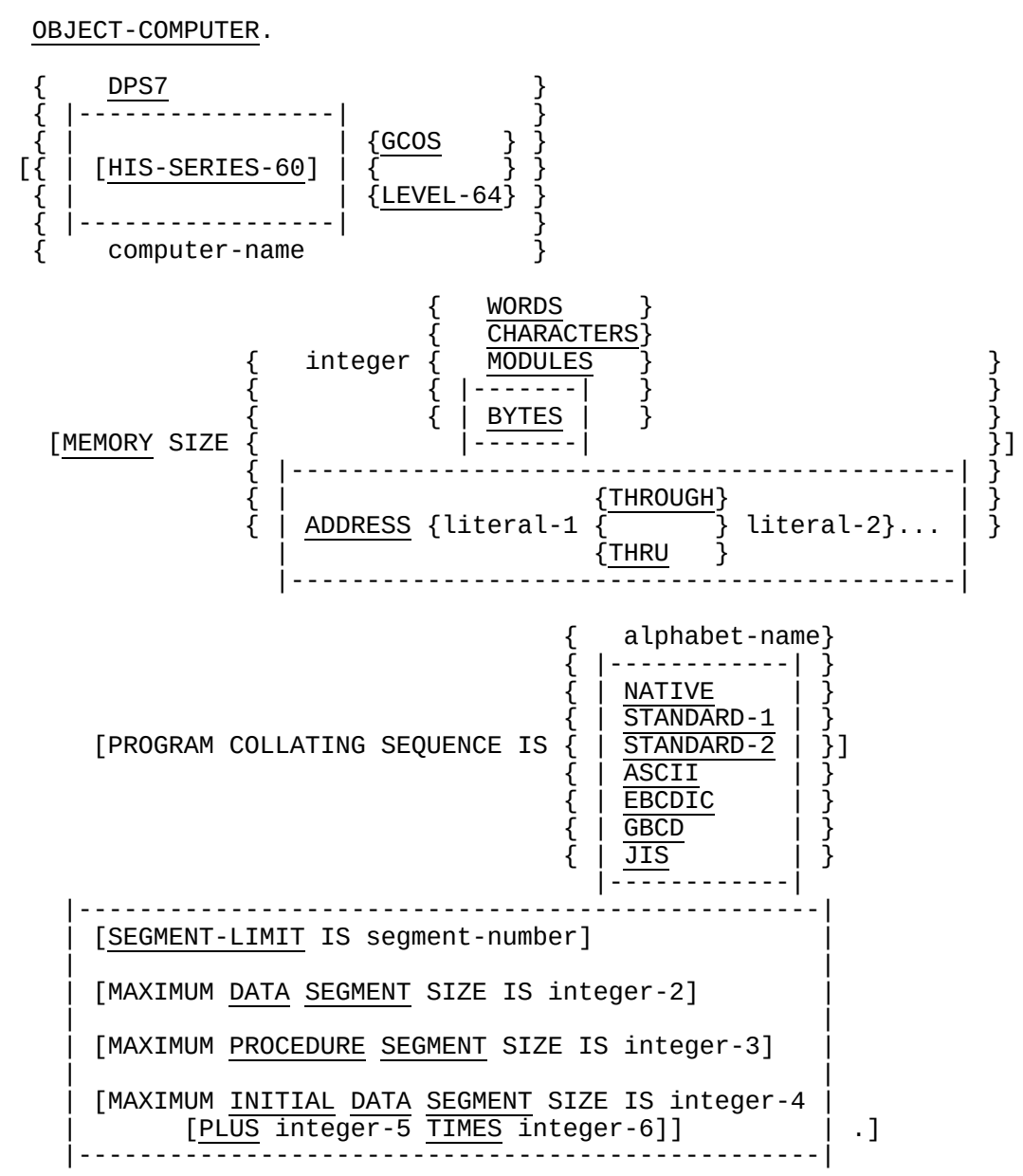
6. The WITH DEBUGGING MODE is not functional until all COPY and REPLACE statements are processed.

7.5 OBJECT-COMPUTER

Description

The OBJECT-COMPUTER paragraph provides a means of describing the computer on which the compiled program is to be executed. The MEMORY SIZE clause and the SEGMENT-LIMIT clause are obsolete elements in Standard COBOL because they are to be deleted from the next revision of Standard COBOL.

Format



Syntax Rules

1. Computer-name may be any user-defined word.

|2. Each literal may be numeric or non-numeric; when numeric, it must be an unsigned integer. Figurative constants are not allowed.|

3. Segment-number must be an integer ranging in value from 1 through 49.

|4. The words THROUGH and THRU are equivalent.

5. Integer-2 must not exceed 4194304. Integer-3 must not exceed 65536. Integer-4 must not exceed 4194304, integer-5 must not exceed 32767, integer-6 must not exceed 4194304.|

General Rules

1. All clauses of the OBJECT-COMPUTER paragraph apply to the program in which they are explicitly or implicitly specified and to any program contained within that program.

2. The MEMORY SIZE clause is given for documentation purposes only. It is accepted for compatibility.

|3. When the OBJECT-COMPUTER paragraph is not specified and the program is contained within a program including an OBJECT-COMPUTER paragraph, the object computer is DPS7.

4. When the OBJECT-COMPUTER paragraph is specified, but the object-computer-entry is not specified, the effect is as though the OBJECT-COMPUTER paragraph is replaced in the source-program with a paragraph of the form:

   OBJECT-COMPUTER. DPS7.|

5. If the PROGRAM COLLATING SEQUENCE clause is specified, the collating sequence |explicitly stated| or associated with alphabet-name is used to determine the truth value of any non-numeric comparisons that are:

   a. Explicitly specified in relation conditions (See "Relation Condition", Chapter 10).

   b. Explicitly specified in condition-name conditions. (See "Condition-Name Condition", Chapter 10).

   c. Implicitly specified by the presence of a CONTROL clause in a Report Description entry (See Chapter 8).

6. If the PROGRAM COLLATING SEQUENCE clause is not specified, the native collating sequence is used |, namely, the EBCDIC collating sequence.

7. For DPS7, EBCDIC and NATIVE are equivalent. However, should the program be executed on another system, there may be a change in the meaning of NATIVE.|

8. If the PROGRAM COLLATING SEQUENCE is specified, the initial program collating sequence is the collating sequence |explicitly stated or| associated with the alphabet-name specified in that clause.

9. The program collating sequence established in the OBJECT-COMPUTER paragraph is applied to any non-numeric merge or sort keys unless the COLLATING SEQUENCE phrase is specified in the respective MERGE or SORT statement (See the "MERGE Statement" Chapter 12 and the "SORT Statement" Chapter 13).

|10. For the meaning of the explicitly stated collating sequences, see the "SPECIAL-NAMES" paragraph, later in this section.

11. The MAXIMUM DATA SEGMENT SIZE clause specifies the approximate size in bytes of the largest data segment which will be created by the compiler for the described data whose length is not greater than the specified size. If this clause is not present, MAXIMUM DATA SEGMENT SIZE IS 4096 is assumed. This clause whether explicit or implicit is overridden by the DSEGMAX option of CBL JCL statement.

12. The MAXIMUM PROCEDURE SEGMENT SIZE clause specifies the approximate size in bytes of the largest procedure segment which will be created by the compiler. If this clause is not present, MAXIMUM PROCEDURE SEGMENT SIZE IS 4096 is assumed. This clause, whether explicit or implicit, is overridden by the PSEGMAX option of the CBL JCL statement.

13. The MAXIMUM INITIAL DATA SEGMENT SIZE clause specifies the maximum size and the number of areas allocated in the stack segment to contain data described in INITIAL programs. Integer-4 specifies the maximum size in bytes of the first or unique area. Integer-5 specifies the maximum number of additional areas. Integer-6 specifies the maximum size in bytes of each additional area. If the clause is not present, MAXIMUM INITIAL DATA SEGMENT SIZE IS 65536 is assumed. This clause whether explicit or implicit is overridden by the ISEGMAX option of the CBL JCL statement.|

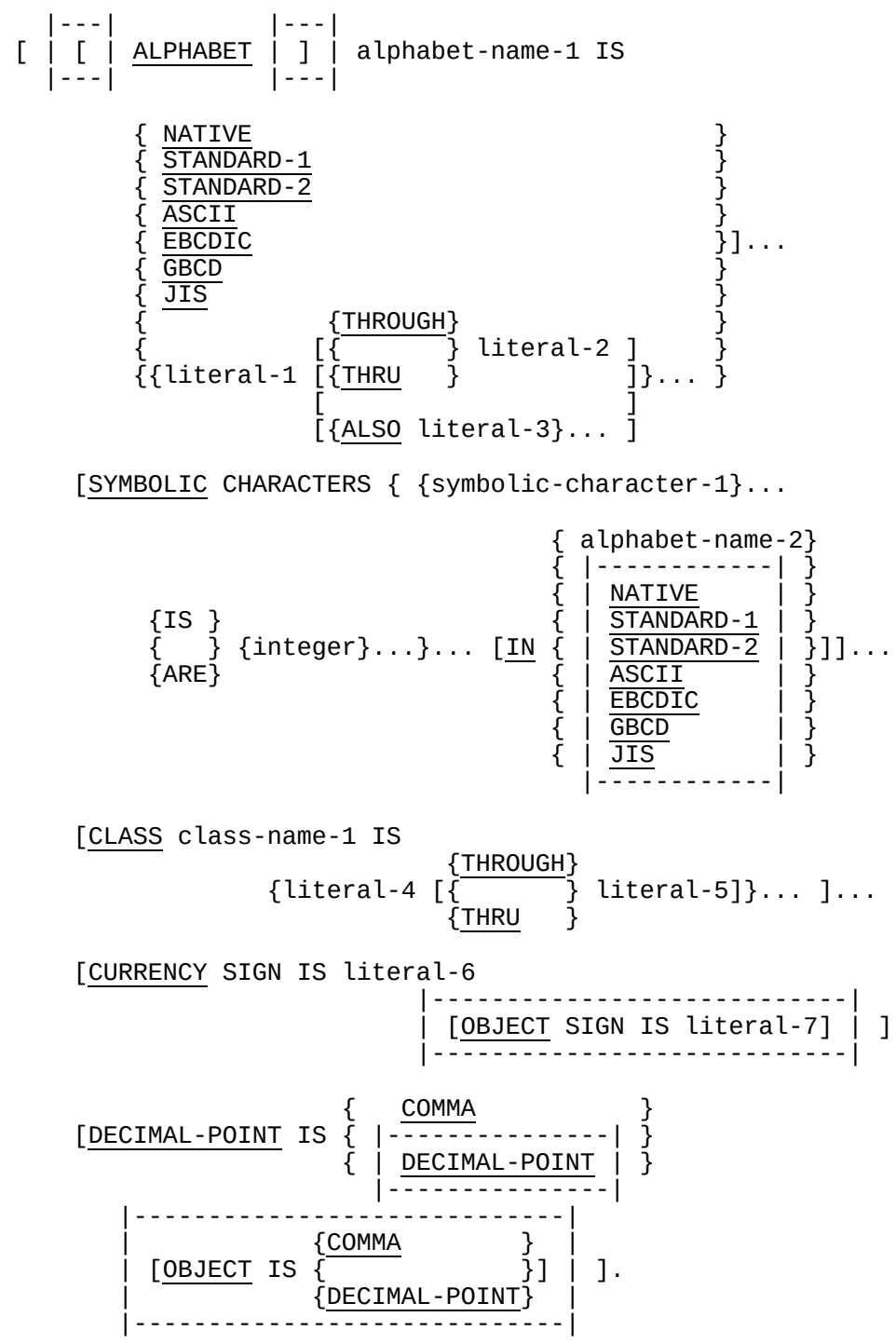
7.6 SPECIAL-NAMES

Description

The SPECIAL-NAMES paragraph provides a means for specifying the currency sign, choosing the decimal point, specifying symbolic characters, relating implementor-names to user-specified mnemonic-names, relating alphabet-names to character sets or collating sequences, and relating class-names to sets of characters.

Format

```
SPECIAL-NAMES

      [SWITCH-n IS mnemonic-name-1            ]
      [    [ON STATUS IS condition-name-1     ]
      [    [OFF STATUS IS condition-name-2]]  ]
      [                                       ]
      [SWITCH-n IS mnemonic-name-1            ]
      [    [OFF STATUS IS condition-name-2    ]
      [    [ON STATUS IS condition-name-1]]   ]...
      [                                       ]
      [SWITCH-n ON STATUS IS condition-name-1 ]
      [    [OFF STATUS IS condition-name-2]   ]
      [                                       ]
      [SWITCH-n OFF STATUS IS condition-name-2]
      [    [ON STATUS IS condition-name-1]    ]

       {LNm }
      [{    } IS mnemonic-name-2]...
       {LN-m}

       [CHANNEL-p IS mnemonic-name-3]...

       {SYSIN  }
      [{       } IS mnemonic-name-4]...
       {SYSIN-q}

       {SYSOUT  }
      [{        } IS mnemonic-name-5]...
       {SYSOUT-q}

       {CONSOLE  }
      [{         } IS mnemonic-name-6]...
       {CONSOLE-q}

    {  {ALTERNATE-CONSOLE  }   }
    {  {ALTERNATE-CONSOLE-q}   }
   [{|-----------------------| } IS mnemonic-name-7]...
    {| {ALTERNATE CONSOLE  } | }
    {| {ALTERNATE CONSOLE-q} | }
     |-----------------------|

       {TERMINAL  }
      [{          } IS mnemonic-name-8]...
       {TERMINAL-q}
```



Syntax Rules

1. "n" in SWITCH-n is an unsigned integer ranging from 0 through 31 and written without leading zeroes.

2. LNm and LN-m are equivalent.

3. "m" in LNm and LN-m is an unsigned integer ranging from 1 through 255 and written without leading zeroes.

4. "p" in CHANNEL-p is an unsigned integer ranging from 1 through 12, and written without leading zeroes.

5. "q" in SYSIN-q, SYSOUT-q, CONSOLE-q, ALTERNATE-CONSOLE-q and TERMINAL-q is the digit 0, 1 or 2 or the letter X.

6. If the literal phrase of the ALPHABET clause is specified, a given character must not be specified more than once in that clause.

7. The literals specified in the literal phrase of the ALPHABET clause:

   a. If numeric, must be unsigned integers and must have a value within the range of one (1) through the maximum number of characters in the native character set (256).

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

8. Literal-1, literal-2, literal-3, literal-4 and literal-5 must not specify a symbolic-character figurative constant.

9. Literal-6 |and literal-7| are non-numeric literals.

10. The words THRU and THROUGH are equivalent.

11. The same symbolic-character-1 must appear only once in a SYMBOLIC CHARACTERS clause.

12. The relationship between each symbolic-character-1 and the corresponding integer-1 is by position in the SYMBOLIC CHARACTERS clause. The first symbolic-character-1 is paired with the first integer-1; the second symbolic-character-1 is paired with the second integer-1; and so on.

13. There must be a one to one correspondence between occurrences of symbolic-character-1 and occurrences of integer-1.

14. The ordinal position specified by integer-1 must exist in the native character set. If the IN phrase is specified, the ordinal position must exist in the character set named by alphabet-name-2.

15. The literals specified in the literal-4 phrase:

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

   b. If non-numeric and associated with a THROUGH phrase, must each be one character in length.

16. Literal-6 |and literal-7 must not be symbolic-character figurative constants|, they may be non symbolic-character figurative constants.|

General Rules

1. All clauses specified in the SPECIAL-NAMES paragraph for a program also apply to programs contained within that program.The condition names specified in the containing program's SPECIAL-NAMES paragraph may be referenced from any contained program.

2. SWITCH-n references an external switch, whose "on" status and/or "off" status may be associated with condition-names. The status of that switch may be interrogated by testing these condition-names (see "Switch-Status Condition", Chapter 10).

|3. SWITCH-n references an external switch whose status may be altered by execution of a SET statement which specifies as its operand the mnemonic-name associated with that switch, or SWITCH-n itself (see the "SET Statement", Chapter 13).|

4. Mnemonic-name-2 may only be specified in the ADVANCING phrase of the WRITE statement. It then specifies that the printer page is advanced to the line whose absolute number in the page is "m" as specified in the related LN-m (or lnm).

5. Mnemonic-name-3 may only be specified in the ADVANCING phrase of the WRITE statement. It then specifies that the printer page is advanced to a position governed by the "p"th channel of the vertical-format unit, "p" being as specified in the related CHANNEL-p.

6. The suffix -q in the SYSIN, SYSOUT, CONSOLE, ALTERNATE-CONSOLE, |ALTERNATE CONSOLE,| or TERMINAL clause specify the form of the data read or written on the external medium when an ACCEPT or DISPLAY statement referencing the special mnemonic-name is executed (See "Definition of a Legible Equivalent", Chapter 3).

7. Mnemonic-name-4 may only be specified in the FROM phrase of the ACCEPT statement. It then specifies that the statement accepts data from the file whose internal file name (ifn) is H_RD.

8. Mnemonic-name-5 may only be specified in the UPON phrase of the DISPLAY statement. It then specifies that the statement displays data on the file whose internal file name (ifn) is H_PR. The data is displayed in SSF format.

9. Mnemonic-name-6 may only be specified in the FROM phrase of the ACCEPT statement and in the UPON phrase of the DISPLAY statement. It then specifies that the statement accepts data from, or displays data upon, the operator console.

10. Mnemonic-name-7 may only be specified in the FROM phrase of the ACCEPT statement and in the UPON phrase of the DISPLAY statement. If the program interactively runs with an IOF terminal, it specifies that the statement accepts data from, or displays data upon, that very console. Otherwise it then specifies that the statement accepts data from, or displays data upon, the alternate operator console specified in the CONSOLE JCL statement. If no alternate console is specified, data is accepted from, or displayed upon, the console from which the job is submitted.

11. Mnemonic-name-8 may only be specified in the FROM phrase of the ACCEPT statement and in the UPON phrase of the DISPLAY statement. If the program interactively runs with an IOF terminal, it specifies that the statement accepts data from, or displays data upon, that very console. Otherwise it specifies that the statement accepts data from, or displays data upon the alternate operator console specified in the CONSOLE JCL statement; if no alternate console is specified, data is accepted from, or displayed upon, the console from which the job is submitted.

12. The ALPHABET clause provides a means for relating a name to a specified character code set and/or collating sequence. When alphabet-name-1 is referenced in the PROGRAM COLLATING SEQUENCE clause (see the "OBJECT-COMPUTER Paragraph") or the COLLATING SEQUENCE phrase of a MERGE or SORT statement (see the "MERGE Statement" Chapter 12 or the "SORT Statement" Chapter 13), the ALPHABET clause specifies a collating sequence. When alphabet-name-1 is referenced in a CODE-SET clause in a file description entry, it 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 X3.4-1977, Code for Information Interchange. If the STANDARD-2 phrase is specified, the character code set identified is the International Version of the ISO 7-bit code defined in International Standard 646, 7-Bit Coded Character Set for Information Processing Interchange. Each character of the standard character set is associated with its corresponding character of the native character set.

    b. If the NATIVE or EBCDIC phrase is specified, it is the EBCDIC character set or collating sequence that is used.

    c. If the GBCD phrase is specified, it is the Honeywell Bull 100/400/600 character set or collating sequence that is used.

    d. If the JIS phrase is specified, the Japanese Industry Standard collating sequence is used.

    e. If the literal phrase is specified, the alphabet-name may not be referenced in a CODE-SET clause (see the "CODE-SET Clause", Chapter 9). The collating sequence identified is that defined according to the following rules:

    Rule 1:
    The value of each literal specifies:

    (i) The ordinal number of a character within the native character set, if the literal is numeric. This value must not exceed the value which represents the number of characters in the native character set.

    (ii) The actual character within the native character set, if the literal is non-numeric. If the value of the non-numeric literal contains multiple characters, each character in the literal, starting with the leftmost character, is assigned successive ascending positions in the collating sequence being specified.

    Rule 2:
    The order in which the literals appear in the ALPHABET clause specifies, in ascending sequence, the ordinal number of the character within the collating sequence being specified.

    Rule 3:
    Any characters within the native collating sequence, which are not explicitly specified in the literal phrase, assume a position, in the collating sequence being specified, greater than any of the explicitly specified characters. The

relative order within the set of these unspecified characters is unchanged from the native collating sequence.

Rule 4:

If the THROUGH phrase is specified, the set of contiguous characters in the native character set beginning with the character specified by the value of literal-1, and ending with the character specified by the value of literal-2, is assigned a successive ascending position in the collating sequence being specified. In addition, the set of contiguous characters specified by a given THROUGH phrase may specify characters of the native character set in either ascending or descending sequence.

Rule 5:

If the ALSO phrase is specified, the characters of the native character set specified by the value of literal-1 and literal-3 are assigned to the same ordinal position in the collating sequence being specified or in the character code set that is used to represent the data, and if alphabet-name-1 is referenced in a SYMBOLIC CHARACTERS clause, only literal-1 is used to represent the character in the native character set.

13. The character that has the highest ordinal position in the program collating sequence specified is associated with the figurative constant HIGH-VALUE, except if this figurative constant is specified as a literal in the SPECIAL-NAMES paragraph. If more than one character has the highest position in the program collating sequence, the last character specified is associated with the figurative constant HIGH-VALUE.

14. The character that has the lowest ordinal position in the program collating sequence specified is associated with the figurative constant LOW-VALUE, except if this figurative constant is specified as a literal in the SPECIAL-NAMES paragraph. 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.

15. When specified as literals in the SPECIAL-NAMES paragraph, the figurative constants HIGH-VALUE and LOW-VALUE are associated with those characters having the highest and lowest positions, respectively, in the native collating sequence.

16. If the IN phrase is not specified, symbolic-character-1 represents the character whose ordinal position in the native character set is specified by integer-1. If the IN phrase is specified, integer-1 specifies the ordinal position of the character that is represented in the character set named by alphabet-name-2.

17. The internal representation of symbolic-character-1 is the internal representation of the character that is represented in the native character set.

18. The CLASS clause provides a means for relating a name to the specified set of characters listed in that clause. Class-name-1 can be referenced only in a class condition. The characters specified by the values of the literals in this clause define the exclusive set of characters of which this class-name-1 consists.

    The value of each literal specifies:

    a. The ordinal number of a character within the native character set, if the literal is numeric. This value must not exceed the value which represents the number of characters in the native character set.

    b. The actual character within the native character set, if the literal is non-numeric. If the value of the non-numeric literal contains multiple characters, each character in the literal is included in the set of characters identified by class-name-1.

19. If the THROUGH phrase is specified, the contiguous characters in the native character set beginning with the character specified by the value of literal-4, and ending with the character specified by the value of literal-5, are included in the set of characters identified by class-name-1. In addition, the contiguous characters specified by a given THROUGH phrase may specify characters of the native character set in either ascending or descending sequence.

20. Literal-6, 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:

    a. digits 0 through 9;

    b. alphabetic characters 'A', 'B', 'C', 'D', 'E', 'L', 'P', 'R', 'S', 'V', 'X', 'Z', or the space;
    c. special characters '*', '+', '-', ',', '.', ';', '(', ')', '"', '=', '/', '|and Horizontal Tabulation'..|

    If this clause is not present, only the currency sign defined in the COBOL character set may be used as the currency symbol in the PICTURE clause.

|21. Literal-7 which appears in the OBJECT SIGN IS literal clause, is used at object time while editing to represent the currency symbol. The literal is limited to a single character. When the OBJECT SIGN clause is not present, but the CURRENCY SIGN clause is present, literal-6 is then used instead of literal-7; if neither clause is present, the dollar sign ($) is then used.|

22. 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. |The clause DECIMAL-POINT IS DECIMAL-POINT means that the function of comma and period are those specified by default; this clause is meaningful when the clause OBJECT IS COMMA is also used.

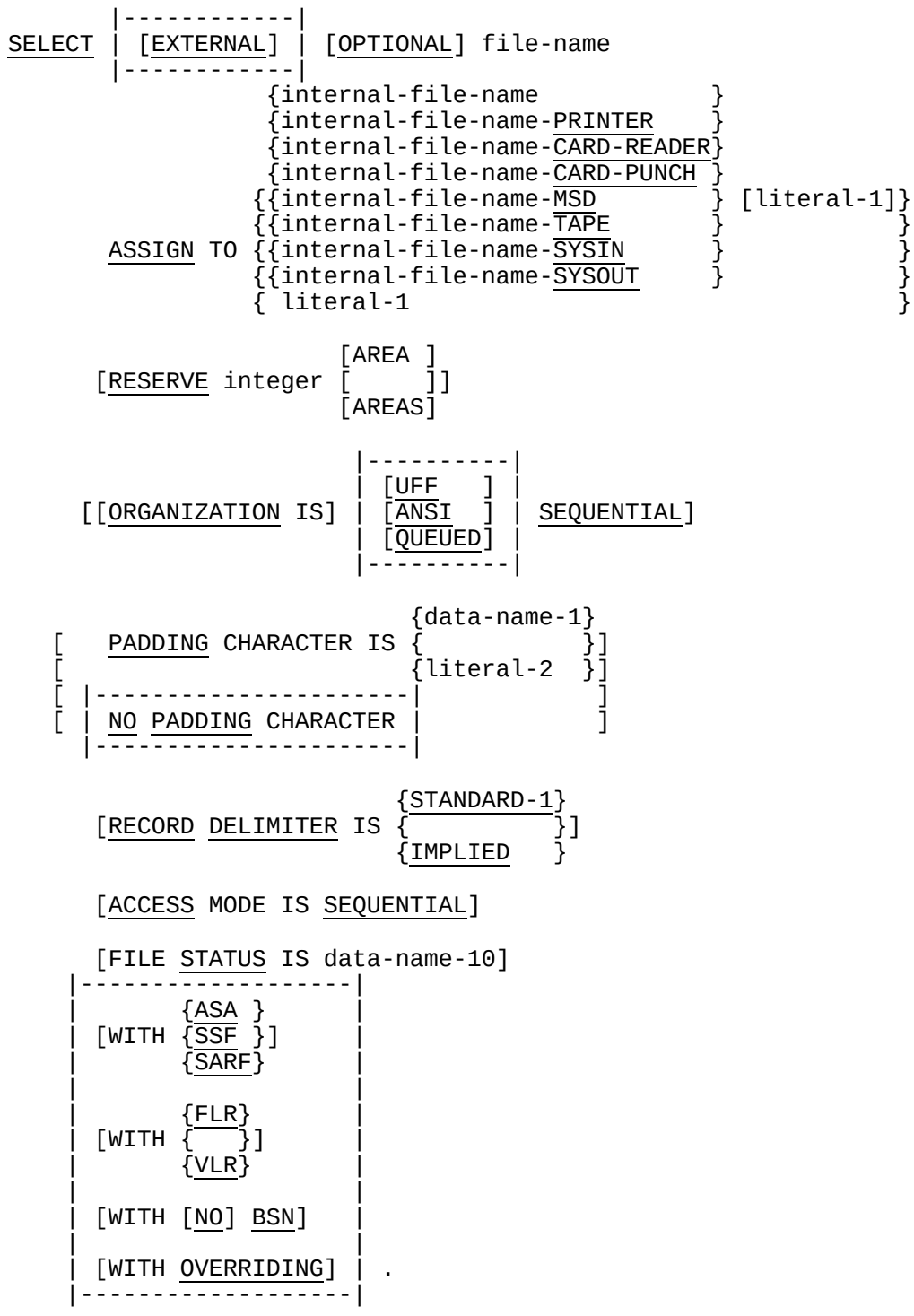
23. The clause OBJECT IS COMMA means that the comma is used at object time while editing to represent the decimal point, and the period to represent the fixed insertion comma. The clause OBJECT IS DECIMAL-POINT means that the period is used at object time while editing to represent the decimal point, and the comma to represent the fixed insertion comma. When neither clause is used, the same characters are used at object time as those specified in the PICTURE clause.|

7.7 FILE-CONTROL-ENTRY

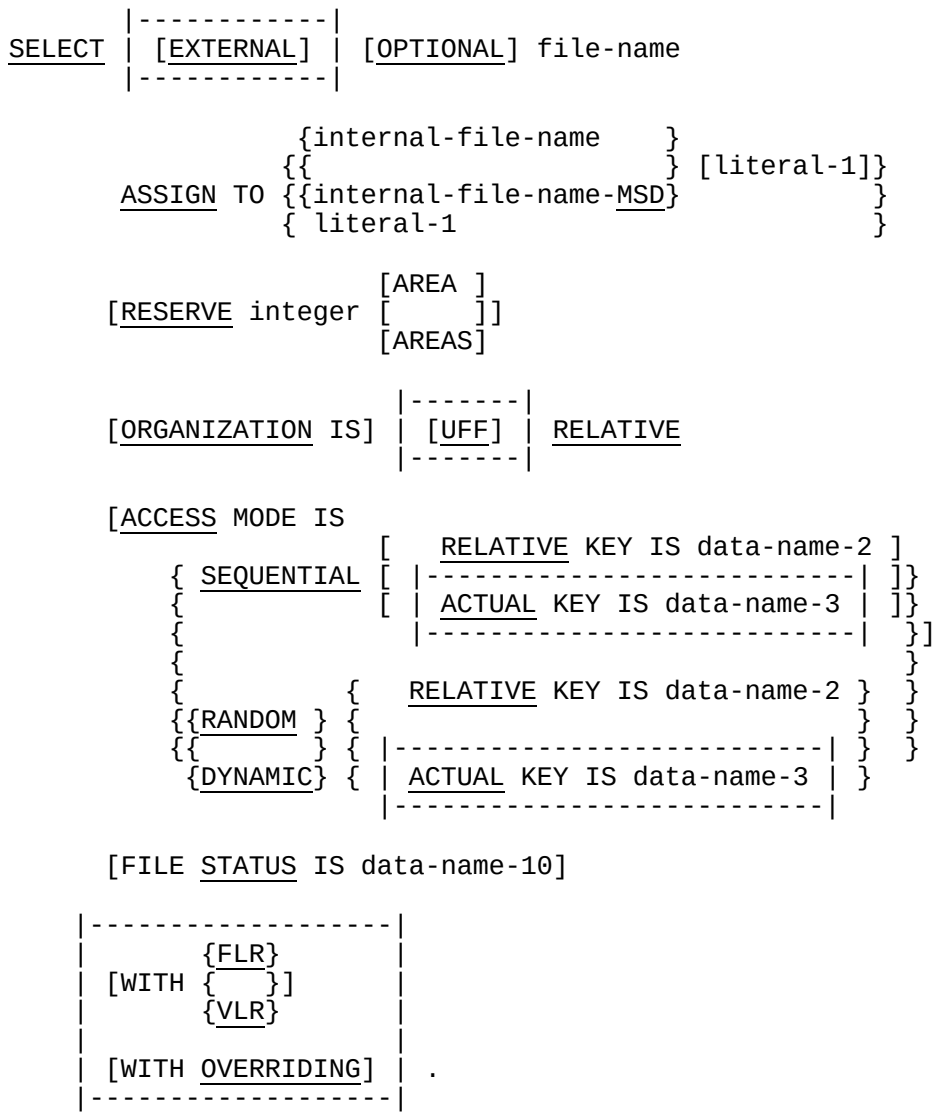
Description

The FILE-CONTROL paragraph allows specification of file-related information.

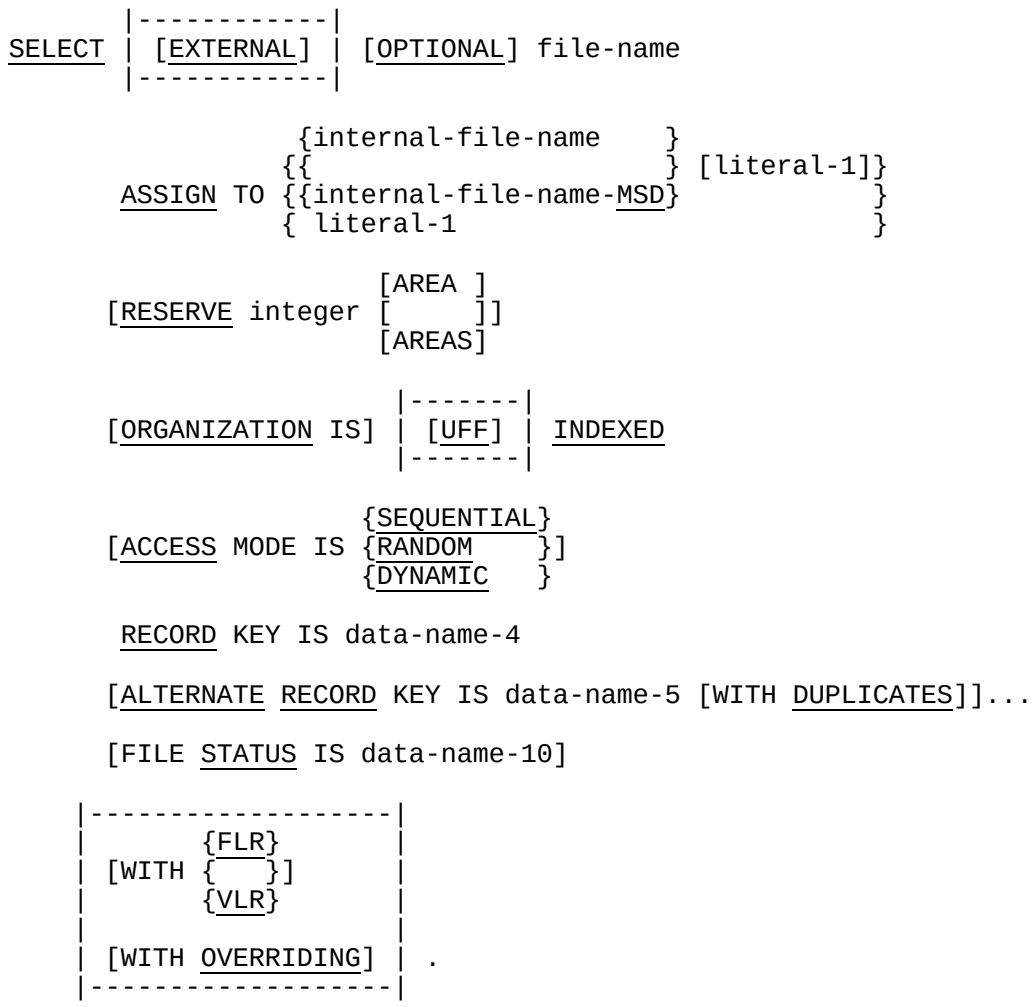
Format 1 (Sequential Files)



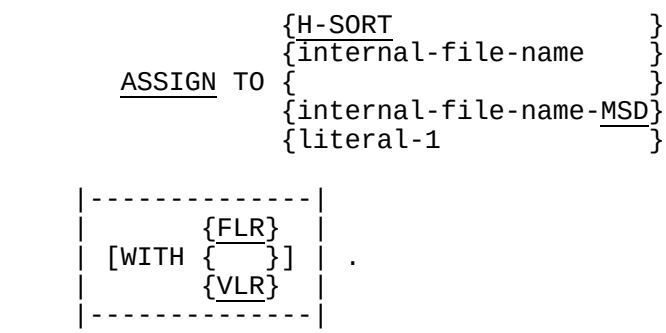
Format 2 (Relative Files)



Format 3 (Indexed Files)



Format 4 (Sort-Merge Files)



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-name in the Data Division must be specified only once in the FILE-CONTROL paragraph. Each file-name specified in a SELECT clause must have a File Description entry or a Sort-Merge File Description entry in the Data Division of the same program.

3. If file-name represents a sort or merge file, the OPTIONAL phrase must not be specified and only the ASSIGN clause |and possibly the WITH FLR or VLR clause| are permitted to follow file-name in the FILE-CONTROL paragraph.

4. Internal-file-name must only consist of letters and digits, and must not exceed 8 characters in length. The only exception to this rule is that the internal-file-name associated to a sort-merge file may be H-SORT, thus containing an hyphen.

   When used, the qualifier following the internal-file-name is adjacent to it, with no intervening space.

   H-SORT is allowed as internal-file-name only for sort-merge files.

   Within a given program, internal-file-names must be unique, except that files referenced in the same MULTIPLE FILE TAPE clause may have the same internal-file-name. In that case the number of characters comprising the internal-file-name must not have a length such that when it is suffixed by the position of the file, it exceeds 8 characters (see the COBOL 85 User's Guide).

5. Literal-1 must be a non-numeric literal and must not be a figurative constant. The contents of literal-1 are described in the following rules; the terms "word" and "separator" are used with a meaning different from that they have when the COBOL text is concerned. This meaning is defined in rules a. through e.:

   a. A word is the concatenation of any positive number of any character except the space, the comma and the equal sign.

   b. There are three types of separators: the separator space, the separator comma and the separator equal.

   c. The character space is a separator space. If more than one consecutive characters space are used, all the consecutive characters space are considered as one separator space.

   d. The character comma is a separator comma. The character comma may be immediately preceded or followed by any number of characters space; in this case, these characters space are not considered as separator space but as part of the separator comma.

   e. The equal sign is a separator equal. The equal sign may be immediately preceded and/or immediately followed by any number of characters space; in this case, these characters space are not considered as separator space but as part of the separator equal.

f. Literal-1 contains either a parameter or a series of parameters. Parameters are defined below in the following rule. In a series of parameters, two consecutive parameters are separated by a separator space or a separator comma. The leftmost parameter may be preceded by the separator space. The rightmost parameter may be followed by the separator space.

g. There are two types of parameters: the positional parameters and the key-word parameters. A positional parameter is a word. A key-word parameter consists in two words separated by a separator equal, the word at the left of the separator equal is the name of this parameter, the word at the right of the separator equal is the value of this parameter.

h. Literal-1 must consist in one or two positional parameters followed by zero, one or more key-word parameters.

i. If there is one positional parameter, it is considered as a file literal. If there are two positional parameters, the first one is considered as an internal-file-name, possibly suffixed as shown in the format of the ASSIGN clause, the second one is a file literal. A file literal must conform to the file description syntax described in the IOF Terminal User's Reference Manual. An internal-file-name must conform to the syntax described above in the previous syntax rules.

j. Key-word parameters names are SHARE, END, ABEND, ACCESS, DENSITY, POOL, FIRSTVOL and LASTVOL. Corresponding allowed values are those described in the JCL Reference Manual.

6. If literal-1 is present and if it is preceded by an internal-file-name, literal-1 must contain no internal-file-name. If literal-1 is present and is not preceded by an internal-file-name, literal-1 must contain an internal-file-name.

7. Data-name-10 may be qualified.

8. Data-name-10 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, Report Section, or Communication Section.

Format 1 (Sequential Files)

9. When the ORGANIZATION clause is specified, but UFF, ANSI and QUEUED are not specified, UFF is implied.

10. Data-name-1 may be qualified.

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

12. Literal-2 must be a one character non-numeric literal.

13. If the STANDARD-1 phrase is specified, the external medium must be a magnetic tape file.

14. The STANDARD-1 phrase may be specified only when the following conditions are met:

    a. The organization of the file is sequential;

    b. The file is not a mass storage file;

    c. The specified internal-file-name is not suffixed or is suffixed with -TAPE;

    d. The CODE-SET clause is explicitly specified for the file as |STANDARD-1 or STANDARD-2| or with an alphabet-name defined as STANDARD-1 or STANDARD-2 in the SPECIAL-NAMES paragraph.

15. The RECORD DELIMITER clause may be specified only for variable length records.

Format 2 (Relative Files)

16. Data-name-2 |and data-name-3| may be qualified.

17. Data-name-2 must be defined as an unsigned integer data item whose description does not contain the PICTURE symbol 'P'.

|18. The data item referenced by data-name-3 is a 5 byte TTRDD type address.|

19. Data-name-2 must not be defined in a Record Description entry associated with that file-name.

20. The ACCESS MODE IS RANDOM clause must not be specified for file-names specified in the USING or GIVING phrase of a SORT or MERGE statement.

21. If a relative file is to be referenced by a START statement, either the RELATIVE KEY phrase, |or the ACTUAL KEY phrase| within the ACCESS MODE clause must be specified for that file.

Format 3 (Indexed Files)

22. The ACCESS MODE IS RANDOM clause must not be specified in the USING or GIVING phrase of a SORT or MERGE statement.

23. Data-name-4 and data-name-5 may be qualified.

24. Neither data-name-4 nor data-name-5 may reference a group data item which has a variable occurrence data item subordinate to it.

25. Data-name-4 must reference a data item of the category alphanumeric within a Record Description entry associated with the file-name to which the RECORD KEY clause is subordinate.

26. Data-name-5 must reference a data item of the category alphanumeric within a Record Description entry associated with the file-name to which the ALTERNATE RECORD KEY clause is subordinate.

27. If the indexed file contains variable length records, the prime record key and each alternate record key must be contained within the first x character positions of the record, where x equals the minimum record size specified for the file (see the "RECORD Clause", Chapter 9).

28. Data-name-5 must not reference an item whose leftmost character position corresponds to the leftmost character position of the prime record key or of any other alternate record key associated with that file.

Format 4 (SORT-MERGE Files)

29. Each sort or merge file described in the Data Division must be specified only once in the FILE-CONTROL paragraph. Each sort or merge file specified in the SELECT clause must have a Sort-Merge File Description entry in the Data Division of the same program.

General Rules

|1. The EXTERNAL phrase causes the file associated with the file-name to be an external file. Programs which are compiled separately, but are part of the same run unit may share files. A shared file must be specified as an external file in the File Control entry for the file in all programs which reference the file. The EXTERNAL phrase must not be specified if the file connector referenced by file-name is an external file connector. (See the "EXTERNAL Clause in FD Entry", Chapter 9.)

2. All rules for input-output concerning the order of operations applied to an external file apply across the independently compiled programs.

3. An external file must have, in each File-Control entry in the run unit, the same values for each of these clauses or phrases: ASSIGN, RESERVE, ORGANIZATION, ACCESS (except for the KEY phrase), ASA, BSN, SSF, SARF, FLR and VLR.|

4. If the file connector referenced by file-name is an external file connector (see the "EXTERNAL FD Entry Clause", Chapter 9), all File Control entries in the run unit which reference this file connector must have:

    a. The same specification for the OPTIONAL phrase.

    b. The same internal-file-name.

    c. A consistent specification of the RECORD DELIMITER clause, namely if the STANDARD-1 phrase is present, they must all have the RECORD DELIMITER STANDARD-1 clause.

    d. The same value for integer-1.

    e. The same organization.

    f. The same specification for the PADDING CHARACTER clause. If data-name-1 is specified, it must reference an external data item.

g. The same Data Description entry for data-name-4 with the same relative location within the associated record.

h. The same Data Description entry for data-name-5, the same relative location within the associated record, the same number of alternate record keys, and the same DUPLICATES phrase.

i. The same access mode.

j. The same external data item for data-name-2 |or data-name-3| in the RELATIVE KEY or |ACTUAL KEY| phrase.

5. The OPTIONAL phrase applies only to files opened in the input, I-O, or extend mode. Its specification is required for files that are not necessarily present each time the program is executed.

6. The ASSIGN clause specifies the association of the file referenced by file-name to the internal-file-name used in the JCL statements to refer to the file. The suffix that is optionally connected by an hyphen to the internal file name, is given for documentation only, except in certain cases specified below under SEQUENTIAL FILES. If literal-1 is specified, it contains an external file-name to which ifn may be assigned. The order of precedence at open time, for file assignment is:

   a. the last executed ASSIGN verb containing the TO FILE phrase referencing the file

   b. then the JCL ASSIGN statement,

   c. then the literal-1 in the ASSIGN clause of the FILE-CONTROL entry.

   If the target file belongs to a Queued file, the file literal in literal-1 must specify the member name except if the QUEUED organization qualifier is specified, in which case, the file literal in literal-1 defines the whole queued file and no member name must appear in it but an ASSIGN statement containing the MEMBER phrase must be executed prior to the OPEN statement.

7. The RESERVE clause allows the user to specify the number of input-output areas allocated. If the RESERVE clause is specified, the number of input-output areas allocated is equal to the value of integer-1. If the RESERVE clause is not specified the number of input-output areas allocated is 2 for the files whose ACCESS MODE IS SEQUENTIAL, 1 otherwise.

8. The ORGANIZATION clause specifies the logical structure of a file. The file organization is established at the time a file is created, is known as the physical file organization and cannot subsequently be changed.

9. When the ORGANIZATION clause is not specified, sequential organization is implied.

10. A disk file whose physical organization is sequential, can be assigned to a file whose SELECT clause specifies that its ORGANIZATION IS RELATIVE provided that overriding is validated explicitly or implicitly and:

    a. the applicable OPEN statements are for INPUT regardless of the ACCESS MODE or,

    b. the applicable OPEN statements are for I-O and the ACCESS MODE IS SEQUENTIAL clause is specified.

    However, the START statement cannot be used unless |an ACTUAL KEY is specified or| the file record format is fixed blocked.

11. A file whose physical organization is relative or indexed, can be assigned to a file-name whose SELECT clause specifies that its ORGANIZATION IS SEQUENTIAL, provided that the applicable OPEN statements are for INPUT or I-O, and provided that overriding is validated explicitly or implicitly.

12. A file whose physical organization is indexed, can be assigned to a file-name whose SELECT clause specifies that its ORGANIZATION IS RELATIVE, provided that the applicable OPEN statements are for INPUT, the ACCESS MODE IS SEQUENTIAL, and overriding is validated explicitly or implicitly.

|13. The Organization qualifier, UFF, ANSI or QUEUED specifies the data management access method expected for the file. UFF specifies that the file is expected to be a UFAS file and ANSI an ANSI file. The organization qualifier QUEUED states that the file is a library member and the member name is to be specified at execution time using an ASSIGN statement that contains the TO MEMBER phrase. In the absence of a qualifier, the file is expected to be a UFAS file, except in the following case:

    The SELECT clause specifies an internal-file-name suffixed by -TAPE and the File Description Entry for file-name, in the Data Division, contains a CODE-SET clause with ASCII, STANDARD-1, or alphabet-name specified as ASCII or STANDARD-1, the file is an ANSI file.|

14. If the ACCESS MODE clause is not specified, sequential access is assumed.

15. 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 "File Status Keys Table", at the end of this chapter).

|16. The WITH FLR and WITH VLR phrases respectively specify whether the record format is fixed length or variable length. If none is specified,| the record format depends on the file description given in the DATA DIVISION:

- If the format 1 of the RECORD clause is used in the FD entry, the file has fixed length records.
- If the format 2 of the RECORD clause is used in the FD entry, the file has variable length records.
- If the format 3 of the RECORD clause is used with the DEPENDING ON phrase in the FD entry, the file has variable length records.
- If no RECORD clause is present or if the format 3 of the RECORD clause is used with no DEPENDING ON phrase in the FD entry, variable length record is assumed if the REPORT clause is present or if at least one of the records specified for the file has subordinate to it an entry containing the OCCURS clause with the DEPENDING ON phrase, or if at least 2 records of different length are specified for the file; otherwise, fixed length records are assumed.

|17. The WITH OVERRIDING clause specifies that the identity of the fixed file attributes of this file to those of the actual file is not checked. WITH OVERRIDING is implicit for a file that has not been created by a COBOL 85 program, or that was not cataloged at the time of creation.|

Format 1 (Sequential Files)

18. The PADDING CHARACTER clause specifies the character which is to be used for block padding on sequential files. During input operations, any portion of a block which exists beyond the last logical record and consists entirely of padding characters will be by-passed. During input operations, a logical record which consists solely of padding characters will be ignored. During output operations, any portion of a block which exists beyond the last logical record will be filled entirely with padding characters.

19. If the PADDING CHARACTER clause is not specified, the creation or recognition of padding characters occurs only if the internal-file-name contains the qualifier - TAPE, or if the ANSI organization is specified. See the default padding character rule, below.

20. Literal-2 or the value of the data item referenced by data-name-1, at the time the OPEN statement which creates the file is executed, is used as the value of the padding character. The padding character is a fixed file attribute.

21. If the CODE-SET clause is specified for the file, conversion of the padding character specified by literal-1 or the content of data-name-1 is established for the file when the file is opened.

22. If the PADDING CHARACTER clause is not specified, the value used for the padding character is that of the character that has the highest ordinal position in the code-set implicitly or explicitly specified for the file.

|23. If the NO PADDING CHARACTER clause is specified, no logical records will be ignored during input operations based on the criteria that they consist solely of padding characters.|

24. The RECORD DELIMITER clause is used to indicate the method of determining the length of a variable length record on the external medium. Any method used will not be reflected in the record area or the record size used within the program.

25. If the STANDARD-1 phrase is specified, the method used for determining the length of a variable record is that specified in American National Standard X3.27-1978, Magnetic Tape Labels and File Structure for Information Interchange, and International Standard 1001 1979, Magnetic Tape Labels and File Structure for Information Interchange.

26. If the IMPLIED phrase is specified, or if the RECORD DELIMITER clause is not specified, the method used for determining the length of a variable length record is that implied by the |explicit or| implicit organization qualifier specified in the ORGANIZATION clause of the file-control entry.

27. At the time of a successful execution of an OPEN statement, the record delimiter is the one specified in the RECORD DELIMITER clause in the File Control entry associated with the file-name specified in the OPEN statement.

28. 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.

29. Among the suffixes which may be connected to the internal-file-name, the following have the specified implication:

    -PRINTER and -SYSOUT imply the SSF attribute |unless the WITH ASA or the WITH SARF phrase is specified.|

    -|SYSIN and -SYSOUT imply that the file may already be in the open mode when an OPEN statement is executed, or may not be in the open mode when a CLOSE statement is executed.|

    -TAPE imply the ANSI organization qualifier when the File Description Entry for the file-name contains a CODE-SET clause with |ASCII, STANDARD-1, or| alphabet-name specified as ASCII or STANDARD-|1, and no other organization qualifier is explicitly specified.|

|30. When the ORGANIZATION IS ANSI SEQUENTIAL clause of the file-control-entry is used, the file is assumed to be in ANSI standard format. The code-set is ASCII, regardless of whether the CODE-SET clause is used or not. No other CODE-SET clause can be used.

31. The WITH ASA clause specifies that the first character of the record will be interpreted as a vertical form command if the file is written to a printing device (see the "WRITE Statement", Chapter 13). It is the user's responsibility to correctly set the first character of the record.

32. The WITH SSF clause specifies that the System Standard Format is applied to a file.

33. The WITH SARF clause specifies that the file is always viewed by the program as a SARF file; thus if the file is actually an SSF file, control records and SSF headers are delivered to the program.

34. If the WITH SSF, WITH ASA and WITH SARF clauses are absent, when an SSF file is read, control records and SSF headers are not sent to the program.

35. The WITH [NO] BSN option specifies that if the file is a tape file, the blocks [do not] contain serial numbers. In the absence of the options, WITH BSN is assumed.|

Format 2 (Relative Files)

36. If 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.

37. If the access mode is random, the value of the RELATIVE |or ACTUAL| KEY data item indicates the record to be accessed.

38. When the access mode is dynamic, records in the file may be accessed sequentially and/or randomly (see "General Rules" above).

39. All records stored in a relative file are uniquely identified by the relative record numbers |or by their addresses on the disk.| 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 one (1), and subsequent logical records have relative record numbers of 2, 3, 4, etc.

40. The data item specified by data-name-2 is used to communicate a relative record number between the user and the Mass Storage Control System (MSCS).

|41. The data item specified by data-name-3 is used to communicate a record disk address between the user and the Mass Storage Control System (MSCS).|

42. The relative key data item associated with the execution of an input-output statement is the data item referenced by data-name-2 in the ACCESS MODE clause.

Format 3 (Indexed Files)

43. 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 |or descending| record key values within a given key of reference.

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

45. When the access mode is dynamic, records in the file may be accessed sequentially and/or randomly (see "General Rules" above).

46. The RECORD KEY clause specifies the prime record key for the file with which this clause is associated. The values of the prime record key must be unique among records of the file. This prime record key provides an access path to records in an indexed file.

47. An ALTERNATE RECORD KEY clause specifies an alternate record key for the file with which this clause is associated. This alternate record key provides an alternate access path to records in an indexed file.

48. The data descriptions of data-name-4 and data-name-5 as well as their relative locations within a record must be the same as that used when the file was created. The number of alternate record keys for the file must also be the same as that used when the file was created.

49. The DUPLICATES phrase specifies that the value of the associated alternate record key may be duplicated within any of the records of the file. If the DUPLICATES phrase is not specified, the value of the associated alternate record key must not be duplicated among any of the records in the file.

|50. An external file must have, in each SELECT clause in the run-unit, the same data description entry for data-name-4, as well as the same relative location within the associated record.|

I-O Status

51. 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 a CLOSE, DELETE, OPEN, READ, REWRITE, START or WRITE 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

52. 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

    "4" indicates Logical Error

    "9" indicates DPS 7 Specific

The meanings of the above indications are as follows:

0 - Successful Completion. The input-output statement was successfully executed.

1 - At End. A sequential READ statement was unsuccessfully executed as a result of one of the following:

a. No next |(or previous one, if the PREVIOUS phrase is used)| logical record exists in the file or an optional file was not present.

b. The relative key data item is not large enough.

2 - Invalid Key. The input-output statement was unsuccessfully executed as a result of one of the following:

a. Sequence Error

b. Duplicate Key

c. No Record Found or Optional File not present.

d. Boundary Violation or Relative Key Data Item not large enough.

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.The permanent error condition remains in effect for all subsequent input-output operations on the file unless a specific mechanism is invoked to correct the permanent error condition.

4 - Logical Error: the execution of the input-output statement was unsuccessful as a result of violating a user-defined limit or because of the state of the file.

9 - DPS 7 Specific. The input-output statement was unsuccessfully executed as a result of a condition such as file not open, etc. 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

53. 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:

a. When Status Key 1 contains a value of "0" indicating a successful completion, Status Key 2 may contain a value indicating further information. These values indicate that:

1. For any input-output statement, the value "0" indicates that no further information is available concerning the input-output operation;

2. For a READ statement, the value "2" indicates that the key value for the current key of reference is equal to the value of that same key in the next record |(or previous one, if the PREVIOUS phrase is used)| within the current key of reference.

3. For a REWRITE or WRITE statement, the value "2" indicates that the record just written created a duplicate key value for at least one alternate record key for which duplicates are allowed.

4. For a READ statement, the value "4" indicates that the length of the record being processed does not conform to the fixed file attributes for that file.

5. For an OPEN statement, the value "5" indicates that the referenced optional file is not present at the time of the successfully executed OPEN statement; if the open mode is I-O or extend, the file has been created.

6. For a CLOSE statement with the NO REWIND, REEL or UNIT , or FOR REMOVAL phrase, or for an OPEN statement with the NO REWIND phrase, the value "7" indicates that the referenced file is a non reel/unit medium.

b. When Status Key 1 contains a value of "1" indicating an at end condition, status key 2 is used to designate the cause of the condition as follows:

1. A value of "0" in status key 2 indicates that a sequential READ statement is attempted and no next logical record exists in the file because the end of the file has been reached, |(or, if the PREVIOUS phrase is used, no previous logical record exists in the file because the beginning of the file has been reached)|, or a sequential READ statement is attempted for the first time on an optional input file that is not present.

2. A value of "4" in status key 2 indicates that a sequential READ statement is attempted for a relative file and the number of significant digits in the relative number is larger than the size of the relative key data item described for that file.

3. A value of "5" in status key 2 indicates that a sequential READ statement is attempted for the first time on an optional input file that is not present.

c. 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 as follows:

1. A value of "1" in status key 2 indicates a sequence error for a sequentially accessed indexed file. The prime 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, or the ascending sequence requirements for successive record key values has been violated (see the "WRITE Statement", Chapter 13).

2. A value of "2" in status key 2 indicates a duplicate key. An attempt has been made to write or rewrite a record that would create a duplicate prime record key or a duplicate alternate record key without the DUPLICATE phrase in an indexed file, or an attempt has been made to write a record that would create a duplicate key in a relative file.

3. A value of "3" in status key 2 indicates that no record has been found. An attempt has been made to access a record identified by a key, and that record does not exist in the file.

4. A value of "4" in status key 2 indicates a boundary violation. An attempt has been made to write beyond the externally-defined boundaries of a relative or indexed file or a sequential WRITE statement has been attempted for a relative file and the number of significant digits in the relative record number is larger than the size of the relative key data item described for that file.

5. A value of "5" in status key 2 indicates that a START, a DELETE, a REWRITER or a random READ statement has been attempted on an optional input file that is not present.

d. When status key 1 contains a value of "3", indicating a permanent error condition, status key 2 is used to designate the cause of that condition as follows:

1. A value of "0" in status key 2 indicates that no further information is available concerning the input-output operation.

2. A value of "4" indicates that an attempt has been made to write beyond the externally-defined boundaries of a sequential file.

3. A value of "5" indicates that an OPEN statement with the INPUT, I-O, or EXTEND phrase is attempted on a non-optional file that is not present.

4. A value of "7" indicates that an OPEN statement is attempted on a file and that file will not support the open mode specified in the OPEN statement; the possible violations are:

a. The EXTEND or OUTPUT phrase is specified but the file will not support write operations.

b. The I-O phrase is specified but the file will not support the input and output operations that are permitted for the organization of that file when opened in the I-O mode.

c. The INPUT phrase is specified but the file will not support read operations.

5. A value of "8" indicates that an OPEN statement is attempted on a file previously closed with lock.

6. A value of "9" indicates that conflict has been detected between the fixed file attributes and the attributes specified for that file in the program.

e. When status key 1 contains a value of "4" indicating a logical error condition, status key 2 is used to designate the cause of that condition as follows:

1. A value of "1" indicates that an OPEN statement is attempted for a file in the open mode.

2. A value of "2" indicates that a CLOSE statement is attempted for a file not in the open mode.

3. A value of "3" indicates that for a mass storage file in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of a DELETE or REWRITE statement was not a successfully executed READ statement.

4. A value of "4" indicates that a boundary violation exists because:

a. An attempt is made to write or rewrite a record that is larger than the largest or smaller than the smallest record allowed by the RECORD clause of the associated file-name, or

b. An attempt is made to rewrite a record to a sequential file and the record is not the same size as the being replaced.

5. A value of "6" indicates that a READ statement is attempted on a file opened in the input or I-O mode and no valid next record |(or previous one, if the PREVIOUS phrase is used)| has been established because:

a. |The preceding START statement was unsuccessful, or|

b. The preceding READ statement was unsuccessful but did not cause an "at end condition", or

c. The preceding READ statement caused an "at end condition".

6. A value of "7" indicates that the execution of a READ or START statement is attempted on a file not opened in the input or I-O mode.

7. A value of "8" indicates:

a. The execution of a random WRITE statement is attempted on a relative or indexed file not opened in the I-O or output mode.

b. The execution of a sequential WRITE statement is attempted on a file not opened in the output or extended mode.

8. A value of "9" indicates that the execution of a DELETE or REWRITE statement is attempted on a file not opened in the I-O mode.

f. When status key 1 contains a value of "9" indicating a DPS 7 specific restriction, status key 2 is used to designate the cause of that condition as follows.

Note that the values of status key 2 marked with a * may only appear when the clause COBOL-74 FOR FILES is used.

VALUE	MEANING
1	An error in Block Serial Number Checking.
2 *	The block just read has a wrong size or the record being rewritten has not the same length as the record it replaces.
3	The return code "BUSY" is got after an attempt to dynamically assign a file not declared as QUEUED during the execution of an OPEN statement referencing that file.
4	The return code "BUSY" is got after an attempt to dynamically assign a file declared as QUEUED during the execution of an OPEN statement referencing that file.
5 *	The maximum record size of the file just opened is not equal to the maximum record size specified in the program.
6 *	An access to the file is attempted though the file is not open.
7 *	The file is already open when an attempt is made to execute an OPEN statement.
9 *	The statement is not allowed after the OPEN option used (e.g. READ cannot be executed for a file opened in OUTPUT).
I*	The statement is not allowed on an optional file that is not present.
J *	The file description, as completed by JCL, IS INCONSISTENT OR IN CONFLICT WITH THE FILE CHARACTERISTICS.
K	An unrecoverable error occurred while printing and one or more pages must be printed again.
L	The function requested is not available.
M	The internal-file-name specified in the ASSIGN TO clause of the SELECT sentence is not assigned to an external file ($ASSIGN statement missing in JCL).
N	The internal-file-name specified in the ASSIGN TO clause of the SELECT sentence is assigned to an unknown external file (through the $ASSIGN JCL statement).
O	The location or the size of the Record Key, or the location or the size or the number of the Alternate Record Key(s), of an indexed file is not that specified in the program.
P	The form of the file does not fit with the ORGANIZATION specified in the program.
Q	A DELETE statement is attempted though the file has been given by JCL the NODELER attribute at allocation time.
R	A $JOB JCL statement has been read.
S	An incomplete record has been read at the physical end of paper tape.

VALUE	MEANING
O	The location or the size of the Record Key, or the location or the size or the number of the Alternate Record Key(s), of an indexed file is not that specified in the program.
P	The form of the file does not fit with the ORGANIZATION specified in the program.
Q	A DELETE statement is attempted though the file has been given by JCL the NODELER attribute at allocation time.
R	A $JOB JCL statement has been read.
S	An incomplete record has been read at the physical end of paper tape.
T	Too large a block has been read and it has been truncated.
U	Too large a record has been read and it has been truncated.
W	An attempt is made to open a file described with an ORGANIZATION clause that specifies QUEUED and the corresponding actual member name designate no existing member (see the "ASSIGN Statement" rules for details on actual member name setting).

All these values of Status Key 2 correspond to abnormal situations leading to an abortion of execution if no declaratives are used.

Valid Combinations of Status Keys 1 and 2

54. The permissible combinations of the values of Status Key 1 and Status Key 2 are shown in the tables below together with the statements and the file organization for which the combination is applicable.

Table 7-1. File Status Keys (1/2)

		STATUS KEY 1-2	MEANING	OPEN	CLOSE	READ	WRITE	REWRITE	DELETE	START
SUCCESS		00	Correct execution	SRI	SRI	SRI	SRI	SRI	SRI	SRI
		00	File not open (2)		SRI					
		00	File already open (2)	SRI						
		02	Duplicate key			I	I	I		
		04	Length inconsistant			SRI				
		05	File absent (1)	S						
		07	Non reel/unit file	S	S					
AT END		10	End of file reached			SRI				
		10	Temporary end of file reached (3)			SRI				
		14	Relative key too large			R				
		15	|File absent (1)|			SRI				
INVALID KEY	SEQUENCE ERROR	21	Key of the recoed has been modified since the last READ					I		
		21	Keys are not submitted in ascending order at creation time				I			
	DUPLICATE KEY	22	The record with the same key already axists				RI	RI		
	NO RECORD FOUND	23	No record with the specified key exists							I
		23	Record has been previously deleted or was not found or			RI		RI	RI	RI
		23	no current record exists or current record has already been updated or deleted					RI	RI	
		25	|File absent (1)|			RI		RI	RI	RI
	BOUDARY VIOLATION	24	The key falls outside the limits of the file				R			
		24	No room in the overflow area, or the key is greater than the largest key of the file or							
		24	Prime area overflow at creation time				I			
		24	Attemp to write beyond the file limits				RI			

Table 7-1. File Status Keys (2/2)

	STATUS KEY 1-2	MEANING	OPEN	CLOSE	READ	WRITE	REWRITE	DELETE	START
PERMANENT ERROR	30	No further information	SRI	SRI	SRI	SRI	SRI	SRI	SRI
	34	Attemp to write beyond file limits				S			
	35	File not present	SRI						
	37	File does not support open mode	SRI						
	38	File closed with lock	SRI						
	39	Attributes conflict	SRI						
LOGICAL ERROR	41	File already open	SRI						
	42	File not open		SRI					
	43	Last statement not successful READ (sequencial access)					SRI	SRI	
	44	Boundary violation				SRI	SRI		
	46	No valid next or previous record			SRI				
	47	File not opened in input or I-O			SRI				SRI
	48	File not opened in output, extend or I-O				SRI			
	49	File not opened in I-O				SRI		SRI	

NOTES: S stands for SEQUENTIAL file

R stands for RELATIVE file

I stands for INDEXED file

(1) OPTIONAL files

(2) EXTERNAL files, SYSIN files, SYSOUT files

(3) EXTERNAL files

Table 7-2. DPS 7000 Specific File Status Keys

STATUS KEY 1-2	MEANING	OPEN	CLOSE	READ	WRITE	RE-WRITE	DELETE	START
91	Block serial number error				S			
92*	Wrong block size			SRI				
93	Dynamically assigned file is busy	SRI						
94	Dynamically assigned file is busy	S						
95*	Wrong record size							
96*	File not open	SRI		SRI	SRI			
97*	File already open		SRI			SRI	SRI	RI
99*	Statement disallowed according to open mode	SRI		SRI	SRI			
9I*	Dummy file	SRI						RI
9J*	Inconsistency in file description					SRI	SRI	RI
9K	Form recovey	SRI			S			
9L	Function not available		SRI	SRI	SRI	SRI	SRI	RI
9M	IFN not assigned	SRI						
9N	External file name unknown	SRI						
9O	Key location or size unexpected	SRI						
9P	File organization unexpected	I						
9Q	Delete not allowed	SRI					I	
9R	Job card has been read			S				
9S	Incomplete record read at physical end of paper tape			S				
9T	Block truncated			S				
9U	Record larger than specified record area			SRI				
9W	Assigned subfile unknown	S						

NOTES:

S stands for SEQUENTIAL file

R stands for RELATIVE file

I stands for INDEXED file

* values which may only arrear if the clause COBOL74 for files is used

7.8 I-O-CONTROL

Description

The I-O-CONTROL paragraph specifies |the input-output techniques,| the points at which rerun is to be established, and the memory area which is to be shared by different files and the location of files on a multiple file reel. The RERUN clause and the MULTIPLE FILE TAPE clause within the I-O-CONTROL paragraph are obsolete elements in Standard COBOL because they are to be deleted from the next revision of Standard COBOL.

Format

```
I-O-CONTROL.

     |-----------------------------------------------|
   [ | [APPLY {NO-SORTED-INDEX ON {file-name-4}..}. ]. |
     |        {OPTIMIZE ON {file-name-5}         }     |
     |-----------------------------------------------|

       [RERUN ON CHECKPOINT-FILE
                  { integer-1 RECORDS}
             EVERY {                  } OF file-name-6]...
                  {          {REEL}   }
                  {[END OF] {     }   }
                            {UNIT}

             [RECORD    ]
       [SAME [SORT      ] AREA
             [SORT-MERGE]

                       FOR file-name-7 {file-name-8}... ]...

       [MULTIPLE FILE TAPE CONTAINS
             {file-name-9 [POSITION integer-2]}... ]... . ]
```

Syntax Rules

1. The order of appearance of the APPLY clauses is immaterial.

2. Any file-name referenced in the I-O-CONTROL paragraph must be specified in the FILE-CONTROL paragraph of the same program.

|3. Each file-name-4 must be a file whose ORGANIZATION IS [UFF] INDEXED.|

|4. Each file-name-5 must be described with an ORGANIZATION clause that does not contain the QUEUED phrase.|

5. A file-name that represents a sort or merge file cannot appear in |an APPLY clause,| a RERUN clause, or a MULTIPLE FILE clause.|

|6. A file-name that represents an external file cannot appear in a RERUN clause or a SAME clause.|

7. The END OF REEL/UNIT clause may only be used if file-name-6 is a sequentially organized file.

8. Only one RERUN clause may be specified for a given file-name-6.

9. File-name-7 and file-name-8 must not reference an external file connector.

10. SORT and SORT-MERGE are equivalent.

11. A file-name that represents a sort or merge file cannot appear in the SAME clause unless the SORT, SORT-MERGE or RECORD clause is used.

12. More than one SAME clause may be included in a program, subject to the following restrictions:

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

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

    c. A file-name that represents a sort or merge file must not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.

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

    e. If a file-name that does not represent a sort or merge file appears in a SAME AREA clause and one or more SAME SORT AREA or SAME SORT-MERGE AREA clauses, all of the files named in that SAME AREA clause must be named in that SAME SORT AREA or SAME SORT-MERGE AREA clause(s).

13. The files referenced in the SAME AREA, SAME RECORD AREA, SAME SORT AREA, or SAME SORT-MERGE AREA clause need not all have the same organization or access.

14. A file-name that represents a report file can appear in a MULTIPLE FILE TAPE clause or in a SAME clause for which the RECORD phrase is not specified.

General Rules

|1. When the APPLY NO-SORTED-INDEX clause is used, the alternate key index is not sorted during the creation of an indexed file.|

|2. When the APPLY OPTIMIZE CLAUSE is used, the fast access UFAS method applies.|

3. When the RERUN... EVERY integer-1 RECORDS clause is used, the rerun information is written on the standard device whenever approximately integer-1 records of the file referenced by file-name-6 have been processed. Other actions take place at this time if the run unit uses files that are monitored by GAC (see the *COBOL 85 User's Guide*). File-name-6 may reference either an input or an output file.

4. When the RERUN...END OF REEL or END OF UNIT clause is used, the rerun information is written on the standard device whenever the end of a reel, or unit, is reached. In addition, normal reel, or unit, closing functions for file-name-6 are performed. File-name-6 may reference either an input or output file.

5. The SAME AREA clause specifies that two or more files referenced by data-name-7, data-name-8, ... that do not represent sort or merge files are to use the same memory area during processing. The area being shared includes all storage areas assigned to the files specified; therefore, it is not valid to have more than one of the files open at the same time. (see Syntax Rule 11.d above).

6. The SAME RECORD AREA clause specifies that two or more files referenced by data-name-7, data-name-8, ... are to use the same memory area for processing of the current logical record. All of the files may be open at the same time. A logical record in the SAME RECORD AREA is considered as a logical record of each opened file open in the output mode whose file-name appears in this SAME RECORD AREA clause and of the most recently read file open in the input mode whose file-name appears in this SAME RECORD AREA clause. This is equivalent to an implicit re-definition of the area, i.e., records are aligned on the leftmost character position explicitly described for the record (i.e. regardless of declared or implied SSF headers).

7. If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file-names must represent a sort or merge file. This clause specifies that storage is shared as follows:

   a. The SAME SORT AREA or SAME SORT-MERGE AREA clause specifies a memory area which will be made available for use in sorting or merging each sort or merge file named. Thus any memory area allocated for the sorting or merging of a sort or merge file is available for reuse in sorting or merging any of the other sort or merge files.

   b. In addition, storage areas assigned to files that do not represent sort or merge files may be allocated as needed for sorting or merging the sort or merge files named in the SAME SORT AREA or SAME SORT-MERGE AREA clause.

c. Files other than sort or merge files do not share the same storage area with each other. If the user wishes these files to share the same storage area with each other, he must also include in the program a SAME AREA or SAME RECORD AREA specifying file-names associated with these files.

d. During the execution of a SORT or MERGE statement that refers to a sort or merge file named in this clause, any non sort or merge files associated with file-names named in this clause must not be in the open mode.

8. The MULTIPLE FILE TAPE clause is required when more than one file shares the same physical reel of tape |unless the relative position of files on a multiple file reel are specified using the FSN parameter in the ASSIGN JCL statement.| Regardless of the number of files on a single reel, only those files that are used in the object program need be specified. If all file-names have been listed in consecutive order, the POSITION phrase need not be given. If any file in the sequence is not listed, the position relative to the beginning of the tape must be given. Not more than one file on the same tape reel may be open at one time.

8. Data Division - Overview

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:

a) That which is contained in files and enters or leaves the internal memory of the computer from a specified input or output device.

b) That which is developed internally and placed into intermediate or working storage, or placed into specific format for output reporting purposes.

c) Constants which are defined by the user.

The Data Division is subdivided into sections. These are the File, Working-Storage, |Constant,| Linkage, Communication, and Report Sections.

The File Section describes the structure of data, sort or merge files. Each file is defined by a File Description entry or a Sort-Merge Description entry and one or more Record Description entries, or by a File Description entry and one or more Report Description entries. Record description entries are written immediately following the File Description entry. When the File Description entry specifies a file to be used as a Report Writer output file, no Record Description entries are permitted for that file. Report Description entries appear in a separate section of the Data Division, the Report Section.

The Working-Storage Section describes records and subordinate 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 Constant Section only describes data items whose values are assigned in the source program and do not change during the execution of the object program.|

The Linkage Section appears in the called program and describes data items that are to be referred to by the calling program and the called program. Its structure is the same as the Working-Storage Section.

The Communication Section defines the data items in the source program that will serve as the interface between The Message Control System (MCS) and the program.

The Report Section describes the content and format of reports that are to be generated.

Format

```
 DATA DIVISION.

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

[WORKING-STORAGE SECTION.

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

|--------------------------------------|
| [CONSTANT SECTION.                   |
|      [77-level-description-entry]    |
|      [                          ]... ] |
|      [record-description-entry  ]    |
|--------------------------------------|

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

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

[REPORT SECTION.
     [report-description-entry
          {report-group-description-entry}... ]... ]
```

NOTE: Within a file-description-entry the record-description-entry is required if the REPORT clause is not specified; it must be omitted if the REPORT clause is specified. (See "File Description", this chapter).

8.1 FILE SECTION

In a COBOL program the File Description entry (FD) and the Sort-Merge File Description entry (SD) represent the highest level of their respective organizations in the File Section. The Sort-Merge File Description (SD) is a special type of file description. The File Section header is followed by a File Description entry consisting of a level indicator (FD, or SD), a file-name and a series of independent clauses. The clauses of a File Description entry specify the size of the logical and physical records, the presence or absence of label records, the names of the data records or reports which comprise the file and finally, the number of lines to be written on a logical printer page. The entry itself is terminated by a period.

An SD File Description gives information about the size and the names of the data records associated with the file to be sorted or merged. The rules for record blocking and internal storage associated to a sort or merge file are peculiar to the SORT and MERGE statements.

The initial value of data items in the File Section is undefined.

8.2 WORKING-STORAGE SECTION

The Working-Storage Section is composed of the section header, followed by Record Description entries and/or description entries for non-contiguous data items.

8.2.1 Non-Contiguous 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 as non-contiguous elementary items. Each of these items is defined in a separate data description entry which begins with special level-number 77.

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

1. level-number 77

2. |optionally,| data-name

3. The PICTURE clause or the USAGE IS INDEX|, COMP-1, COMP-2, COMP-9, COMP-10, COMP-15 or POINTER| clause.

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

8.2.2 Working-Storage Records

Data elements in Working-Storage which bear a definite hierarchical relationship to one another must be grouped into records according to the rules for formation of Record Descriptions. Data elements in the Working-Storage Section which bear no hierarchical relationship to any other data item may be described as records which are single elementary items. All clauses which are used in Record Descriptions in the File Section can be used in Record Descriptions in the Working-Storage Section.

8.2.3 Working-Storage

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 or any data item not associated with a VALUE clause is undefined.

8.3 CONSTANT SECTION

|The Constant Section is exactly like the Working-Storage Section except that:

1. All data items in the Constant Section must have a VALUE clause.

   (Note that reference is made here not to every data "description", but to every data "item". This allows for the use of the REDEFINES and RENAMES clauses).

2. The data items of the Constant Section may be referenced only where literals may be referenced, i.e., their contents may not be altered during program execution.|

8.4 LINKAGE SECTION

The Linkage Section in a program is meaningful 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, |or if the program uses based data item descriptions.|

Record descriptions or non-contiguous data item descriptions whose names are referenced in the USING phrase of the PROCEDURE DIVISION header and their re-definitions define parameters which are passed to the program when it is called. |Data Descriptions in the Linkage Section that do not define parameters, define based data items.|

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 Record Description entries, and/or data description entries for non-contiguous data items.

8.4.1 Parameters

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. The mechanism by which a correspondence is established between the data items described in the Linkage Section of a called program and data items described in the calling program is described elsewhere in this manual (see "Procedure Division Header" in Chapter 10 and the "CALL Statement" in Chapter 11). 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 must not be associated with data items defined in the Report Section of the calling program. If a data item in the Linkage Section is accessed in a program which is not a called program, the effect is undefined |unless the OPTIONS parameter is used in the STEP JCL statement used to run the object program.|

8.4.2 Based Data Items

|A based data item is not to be allocated but its description can be mapped on any storage. The correspondence between a based data item and the mapped area is established by the execution of a SET statement (see the "SET Statement", Chapter 13). In the case of index-names, no such correspondence is established. Index-names associated with a based record description and index-names associated with data items allocated in the mapped area always refer to separate indices. Any reference to a based data item or to its subordinate data items must be chronologically preceded by the execution of a SET statement that defines a valid address for that based data item.|

8.4.3 Non-Contiguous Linkage Storage

Items in the Linkage Section which bear no hierarchical relationship to one another need not be grouped into records and are classified and defined as non-contiguous elementary items. Each of these 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:

1. level-number 77

2. |optionally,| data-name

3. the PICTURE clause or the USAGE IS INDEX|, COMP-1, COMP-2, COMP-9, COMP-10, COMP-15 or POINTER clause|

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

8.4.4 Linkage Records

Data elements in the Linkage Section which bear a definite hierarchical relationship to one another must be grouped into records according to the rules for formation of Record Descriptions. Data elements in the Linkage Section which bear no hierarchical relationship to any other data item may be described as records which are single elementary items.

8.4.5 Initial Values

The VALUE clause must not be specified in the Linkage Section except in condition-name entries (level 88).

8.5 COMMUNICATION SECTION

In a COBOL program the Communication Description entries (CD) represent the highest level of organization in the Communication Section. The Communication Section header is followed by a Communication Description entry consisting of a level indicator (CD), a cd-name and a series of independent clauses. For input these clauses indicate the queues and sub-queues, the message date and time, the source, the text length, the status and end keys, and the message count. For output these clauses specify the destination count, the text length, the status and error keys, and destinations. The entry itself is terminated by a period. These record areas may be implicitly re-defined by user-specified Record Description entries following the various Communication Description clauses.

8.6 REPORT SECTION

In the Report Section the description of each report must begin with a Report Description entry (RD entry) and be followed by the entries that describe the report groups within the report.

8.6.1 Report Description Entry

In addition to naming the report, the Report Description entry defines the format of each page of the report by specifying the vertical boundaries of the region within which each type of the report group may be printed. The Report Description entry also specifies the control data items. When the report is produced, changes in the values of the control data items cause the detail information of the report to be processed in groups called control groups.

Each report named in the REPORTS clause of a File Description entry in the File Section must be the subject of a Report Description entry in the Report Section. Furthermore each report in the Report Section must be named in one and only one File Description entry.

8.6.2 Report Group Description Entry

The report groups that will comprise the report are described following the Report Description entry. The description of each report group begins with a Report Group Description entry; that is an entry that has a 01 level-number and a TYPE clause. Subordinate to the Report Group Description entry, there may appear group and elementary entries that further describe the characteristics of the report group.

8.7 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 the data name or FILLER clause, if specified, followed by a series of independent clauses as required. A Record Description may have 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 and the elements allowed in a Record Description Entry are explained under the appropriate paragraphs. (See "Concepts of Levels" and "Intra-Record Data Structures", Chapter 3 and "Data Description Complete Entry Skeleton", this chapter).

8.8 FILE DESCRIPTION

DESCRIPTION:

The File Description entry furnishes information concerning the physical structure, identification, and record names or report-names pertaining to a given file.

Format 1 (Sequential Non-Report File)

```
FD file-name

  [IS EXTERNAL]

  [IS GLOBAL]

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

          {CONTAINS integer-3 CHARACTERS                          }
          {                                                       }
          {          |---|              |---|                     }
          {CONTAINS | [ | integer-8 TO | ] | integer-9 CHARACTERS}
          {          |---|              |---|                     }
  [RECORD {             |---------------------------|             }]
          {             | [DEPENDING ON data-name-1] |             }
          {             |---------------------------|             }
          {                                                       }
          {IS VARYING IN SIZE [[FROM integer-4] [TO integer-5]    }
          {     CHARACTERS] [DEPENDING ON data-name-1]            }

        {RECORD IS  } {STANDARD}
  [LABEL {          } {        }]
        {RECORDS ARE} {OMITTED }

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

       {RECORD IS  }
  [DATA {          } {data-name-3}... ]
       {RECORDS ARE}

            {data-name-4}                          {data-name-5}
  [LINAGE IS {          } LINES [WITH FOOTING AT {          }]
            {integer-5  }                          {integer-6  }

                  {data-name-6}                    {data-name-7}
    [LINES AT TOP {          }] [LINES AT BOTTOM {          }]]
                  {integer-7  }                    {integer-8  }

                {   alphabet-name}
                { |------------| }
                { | NATIVE     | }
                { | STANDARD-1 | }
   [CODE-SET IS { | STANDARD-2 | }].
                { | ASCII      | }
                { | EBCDIC     | }
                { | GBCD       | }
                { | JIS        | }
                  |------------|
```

Format 2 (Sequential Report File)

```
FD file-name

      [IS EXTERNAL]

      [IS GLOBAL]

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

              {CONTAINS integer-3 CHARACTERS                }
      [RECORD {                                             }]
              {CONTAINS integer-6 TO integer-7 CHARACTERS}

             {RECORD IS  } {STANDARD}
      [LABEL {           } {         }]
             {RECORDS ARE} {OMITTED }

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

                    {   alphabet-name}
                    { |------------| }
                    { | NATIVE     | }
                    { | STANDARD-1 | }
      [CODE-SET IS  { | STANDARD-2 | }]
                    { | ASCII      | }
                    { | EBCDIC     | }
                    { | GBCD       | }
                    { | JIS        | }
                      |------------|

      {REPORT IS  }
      {           } {report-name-1}... .
      {REPORTS ARE}
```

Format 3 (Relative File)

```
FD file-name

  [IS EXTERNAL]

  [IS GLOBAL]

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

          {CONTAINS integer-3 CHARACTERS                        }
          {                                                     }
          {          |---|              |---|                   }
          {CONTAINS  | [ | integer-8 TO | ] | integer-9         }
          {          |---|              |---|          CHARACTERS }
  [RECORD {            |-----------------------------|          }]
          {            | [DEPENDING ON data-name-1] |          }
          {            |-----------------------------|          }
          {                                                     }
          {IS VARYING IN SIZE [[FROM integer-4] [TO integer-5]  }
          {     CHARACTERS] [DEPENDING ON data-name-1]          }

         {RECORD IS  } {STANDARD}
  [LABEL {           } {         }]
         {RECORDS ARE} {OMITTED  }

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

        {RECORD IS  }
  [DATA {           } {data-name-3}... ]
        {RECORDS ARE}

                {   alphabet-name}
                { |------------| }
                { | NATIVE     | }
                { | STANDARD-1 | }
  [CODE-SET IS  { | STANDARD-2 | }] .
                { | ASCII      | }
                { | EBCDIC     | }
                { | GBCD       | }
                { | JIS        | }
                  |------------|
```

Format 4 (Indexed File)

```
FD file-name

  [IS EXTERNAL]

  [IS GLOBAL]

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

          {CONTAINS integer-3 CHARACTERS                          }
          {                                                       }
          {          |---|              |---|                     }
          {CONTAINS  | [ | integer-8 TO | ] | integer-9           }
          {          |---|              |---|        CHARACTERS   }
  [RECORD {             |-----------------------------|           }]
          {             | [DEPENDING ON data-name-1]  |           }
          {             |-----------------------------|           }
          {                                                       }
          {IS VARYING IN SIZE [[FROM integer-4] [TO integer-5]    }
          {     CHARACTERS] [DEPENDING ON data-name-1]            }

        {RECORD IS  } {STANDARD}
  [LABEL {          } {        }]
        {RECORDS ARE} {OMITTED }

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

       {RECORD IS  }
  [DATA {          } {data-name-3}... ]
       {RECORDS ARE}

                {   alphabet-name}
                { |------------| }
                { | NATIVE     | }
                { | STANDARD-1 | }
  [CODE-SET IS  { | STANDARD-2 | }] .
                { | ASCII      | }
                { | EBCDIC     | }
                { | GBCD       | }
                { | JIS        | }
                  |------------|
```

Syntax Rules

All Formats

1. The level indicator FD identifies the beginning of a File Description entry and must precede the file-name.

2. The clauses which follow file-name may appear in any order.

Formats 1 and 2

3. File-name may only reference a sequential file.

Formats 1, 3 and 4

4. One or more Record Description entries must follow the File Description entry.

Format 2

5. Each report named in the REPORT clause must be the subject of a Report Description entry in the Report Section.

6. No Record Description entries which define data records may follow the File Description entry.

Format 3

7. File-name may only reference a relative file.

Format 4

8. File-name may only reference an indexed file.

General Rules

1. A File Description entry associates file-name with a file connector.

2. If the File Description entry for a sequential file contains the LINAGE clause and the EXTERNAL clause, the LINAGE-COUNTER data item is an external data item. If the File Description entry for a sequential file contains the LINAGE clause and the GLOBAL clause, the special register LINAGE-COUNTER is a global name.

|3. When the file is external, the maximum record length implied by the Record Description Entries subordinate to the File Description Entry must be the same in each File Description Entry in the run-unit.|

8.9 SORT-MERGE FILE DESCRIPTION- COMPLETE ENTRY SKELETON

Description

The Sort-merge File Description furnishes information concerning the physical structure and record-names pertaining a sort or merge file.

Format

```
                |-------------|
SD file-name    | [IS GLOBAL] |
                |-------------|

           {CONTAINS integer-4 CHARACTERS                          }
           {                                                       }
           {          |---|               |---|                   }
           {CONTAINS  | [ | integer-1 TO  | ] | integer-2         }
           {          |---|               |---|         CHARACTERS }
  [RECORD  {             |-----------------------------|           }]
           {             | [DEPENDING ON data-name-1]  |           }
           {             |-----------------------------|           }
           {                                                       }
           {IS VARYING IN SIZE [[FROM integer-3] [TO integer-4]    }
           {     CHARACTERS] [DEPENDING ON data-name-1]            }

           {RECORD IS  }
    [DATA  {           } {data-name-2}... ].
           {RECORDS ARE}
```

Syntax Rules

1. The level indicator SD identifies the beginning of the Sort-merge file Description entry and must precede the file-name.

2. The clauses that follow the name of the file are optional and their order of appearance is immaterial.

3. One or more record description entries must follow the Sort-merge file Description entry; however, no input-output statements may be executed for this file.

8.10 COMMUNICATION DESCRIPTION - COMPLETE ENTRY SKELETON

Description

The Communication Description specifies the interface area between the Message Control System (MCS) and a COBOL PROGRAM.

Format 1

```
CD cd-name FOR [INITIAL] INPUT

    |-------------|
    | [IS GLOBAL] |
    |-------------|

    [[[SYMBOLIC QUEUE IS data-name-1]                ]
    [                                                ]
    [       [SYMBOLIC SUB-QUEUE-1 IS data-name-2]    ]
    [                                                ]
    [       [SYMBOLIC SUB-QUEUE-2 IS data-name-3]    ]
    [                                                ]
    [       [SYMBOLIC SUB-QUEUE-3 IS data-name-4]    ]
    [                                                ]
    [       [MESSAGE DATE IS data-name-5]            ]
    [                                                ]
    [       [MESSAGE TIME IS data-name-6]            ]
    [                                                ]
    [       [SYMBOLIC SOURCE IS data-name-7]         ]
    [                                                ]
    [       [TEXT LENGTH IS data-name-8]             ].
    [                                                ]
    [       [END KEY IS data-name-9]                 ]
    [                                                ]
    [       [STATUS KEY IS data-name-10]             ]
    [                                                ]
    [       [MESSAGE COUNT IS data-name-11]]         ]
    [                                                ]
    [ [data-name-1 data-name-2 data-name-3           ]
    [       data-name-4 data-name-5 data-name-6      ]
    [       data-name-7 data-name-8 data-name-9      ]
    [       data-name-10 data-name-11]               ]
```

Format 2

```
CD cd-name FOR OUTPUT

    |-------------|
    | [IS GLOBAL] |
    |-------------|

      [DESTINATION COUNT IS data-name-1]

      [TEXT LENGTH IS data-name-2]

      [STATUS KEY IS data-name-3]

      [DESTINATION TABLE OCCURS integer-1 TIMES
           [INDEXED BY {index-name-1}... ]]

      [ERROR KEY IS data-name-4]

      [SYMBOLIC DESTINATION IS data-name-5].
```

Format 3

```
CD cd-name FOR [INITIAL] I-O

    |-------------|
    | [IS GLOBAL] |
    |-------------|

    [[[MESSAGE DATE IS data-name-1]              ]
    [                                            ]
    [      [MESSAGE TIME IS data-name-2]         ]
    [                                            ]
    [      [SYMBOLIC TERMINAL IS data-name-3]    ]
    [                                            ]
    [      [TEXT LENGTH IS data-name-4]          ] .
    [                                            ]
    [      [END KEY IS data-name-5]              ]
    [                                            ]
    [      [STATUS KEY IS data-name-6]]          ]
    [                                            ]
    [ [data-name-1 data-name-2 data-name-3       ]
    [       data-name-4 data-name-5 data-name-6] ]
```

Syntax Rules

All Formats

1. A CD entry must appear only in the Communication Section.

Formats 1 and 3

2. Within a single program, the INITIAL clause may be specified in only one CD entry. The INITIAL clause must not be used in a program that specifies the USING phrase of the Procedure Division Header. (See the "Procedure Division Header", Chapter 10.)

3. Except for the INITIAL clause, the optional clauses may be written in any order.

4. If neither option for specifying the interface area is used, a level 01 Data Description entry must follow the CD entry.

Either option may be followed by a level 01 Data Description entry.

Format 1

5. Record Description entries following an input CD implicitly re-define the record area established by the input CD entry and must describe a record of exactly 87 Standard Data Format characters. Multiple re-definitions of this record are permitted; however, only the first re-definition may contain VALUE clauses. The Message Control System (MCS) always references the record according to the data description defined in general rule 2 (see the "VALUE Clause", Chapter 9).

6. Data-name-1, data-name-2, ..., data-name-11 must be unique within the CD. Within this series, any data-name may be replaced by the reserved word FILLER. |Ending consecutive FILLER's may be omitted in a data-name series that follows the mandatory INPUT clause.|

Format 2

7. The optional clauses may be written in any order.

8. If none of the optional clauses of the CD is specified, a level 01 Data Description entry must follow the CD entry.

9. Record Description entries subordinate to an output CD entry implicitly re-define the record area established by the output CD entry. Multiple re-definitions of this record are permitted; however, only the first re-definition may contain VALUE clauses. The Message Control System (MCS) always references the record according to the Data Description defined in General Rule 16 (see the "VALUE Clause", Chapter 9).

10. Data-name-1, data-name-2, ... , data-name-5 must be unique within a CD entry.

11. If the DESTINATION TABLE OCCURS clause is not specified, one (1) ERROR KEY and one (1) SYMBOLIC DESTINATION area are assumed. In this case, subscripting is not permitted when referencing these data items.

12. If the DESTINATION TABLE OCCURS clause is specified, data-name-4 and data-name-5 may be referenced only by subscripting.

Format 3

13. Record Description entries following an input-output CD entry implicitly re-define the record area established by the input-output CD entry and must describe a record of exactly 33 Standard Data Format characters. Multiple re-definitions of this record are permitted; however, only the first re-definition may contain VALUE clauses. The Message Control System (MCS) always references the record according to the Data Description defined in General Rule 24 (see the "VALUE Clause", Chapter 9).

14. Data-name-1, data-name-2, ... , data-name-6 must be unique within the CD entry. Within this series, any data-name may be replaced by the reserved word FILLER. |Ending consecutive FILLER's may be omitted in a data-name series that follows the mandatory I-O clause.|

General Rules

Format 1

1. The input CD information constitutes the communication between the Message Control System (MCS) and the program, about he message to be handled. This information does not come from the terminal as part of the message.

2. For each input CD entry, a record area of 87 contiguous character positions is allocated. This record area is defined to the Message Control System as follows:

    a. The SYMBOLIC QUEUE clause defines data-name-1 as the name of an elementary alphanumeric data item of 12 characters occupying positions 1 through 12 in the record.

    b. The SYMBOLIC SUB-QUEUE-1 clause defines data-name-2 as the name of an elementary alphanumeric data item of 12 characters occupying positions 13 through 24 in the record.

    c. The SYMBOLIC SUB-QUEUE-2 clause defines data-name-3 as the name of an elementary alphanumeric data item of 12 characters occupying positions 25 through 36 in the record.

    d. The SYMBOLIC SUB-QUEUE-3 clause defines data-name-4 as the name of an elementary alphanumeric data item of 12 characters occupying positions 37 through 48 in the record.

    e. The MESSAGE DATE clause defines data-name-5 as the name of a data item whose implicit description is that of an integer of 6 digits without an operational sign occupying character positions 49 through 54 in the record.

    f. The MESSAGE TIME clause defines data-name-6 as the name of a data item whose implicit description is that of an integer of 8 digits without an operational sign occupying character positions 55 through 62 in the record.

g. The SYMBOLIC SOURCE clause defines data-name-7 as the name of an elementary alphanumeric data item of 12 characters occupying positions 63 through 74 in the record.

h. The TEXT LENGTH clause defines data-name-8 as the name of an elementary data item whose implicit description is that of an integer of 4 digits without an operational sign occupying character positions 75 through 78 in the record.i. The END KEY clause defines data-name-9 as the name of an elementary alphanumeric data item of 1 character occupying position 79 in the record.

i. The STATUS KEY clause defines data-name-10 as the name of an elementary alphanumeric data item of 2 characters occupying positions 80 and 81 in the record.

j. The MESSAGE COUNT clause defines data-name-11 as the name of an elementary data item whose implicit description is that of an integer of 6 digits without an operational sign occupying character positions 82 through 87 in the record.

The second option may be used to replace the above clauses by a series of data-names which, taken in order, correspond to the data-names defined by these clauses.

Use of either option results in a record whose implicit description is equivalent to the following:

IMPLICIT DESCRIPTION	COMMENT	
01 data-name-0.		
02 data-name-1	PICTURE X(12).	SYMBOLIC QUEUE
02 data-name-2	PICTURE X(12).	SYMBOLIC SUB-QUEUE-1
02 data-name-3	PICTURE X(12).	SYMBOLIC SUB-QUEUE-2
02 data-name-4	PICTURE X(12).	SYMBOLIC SUB-QUEUE-3
02 data-name-5	PICTURE 9(06).	MESSAGE DATE
02 data-name-6	PICTURE 9(08).	MESSAGE TIME
02 data-name-7	PICTURE X(12).	SYMBOLIC SOURCE
02 data-name-8	PICTURE 9(04).	TEXT LENGTH
02 data-name-9	PICTURE X.	END KEY
02 data-name-10	PICTURE XX.	STATUS KEY
02 data-name-11	PICTURE 9(06).	MESSAGE COUNT

In the above, the information given under 'COMMENT' is for clarification and is not part of the Data Description.

3. The contents of the data items referenced by data-name-2, data-name-3 and data-name-4, when not being used, must contain spaces.

4. The data-items referenced by data-name-1, data-name-2, data-name-3 and data-name-4 contain symbolic names designating queues, sub-queues, ..., respectively. These symbolic names must follow the rules for the formation of system-names, and must have been previously defined to the Message Control System (MCS).

5. A RECEIVE statement causes the serial return of the next message or portion of a message from the queue as specified by the entries of the CD.

   At the time of execution of a RECEIVE statement, the input CD area must contain, in the contents of data-name-1, the name of a symbolic queue. The data items specified by data-name-2, data-name-3 and data-name-4 may contain symbolic sub-queue names or spaces. When a given level of the queue structure is specified, all higher levels must also be specified. If less than all the levels of the queue hierarchy are specified, the MCS determines the next message or portion of a message to be accessed within the queue and/or sub-queue specified in the input CD.

   After the execution of a RECEIVE statement, the contents of the data items referenced by data-name-1 through data-name-4 will contain the symbolic names of all the level of the queue structure.

6. Whenever a program is scheduled by the Message Control System (MCS) to process a message, that program establishes a run unit, and the symbolic names of the queue structure that demanded this activity will be placed in the data items referenced by data-name-1 through data-name-4 of the CD associated with the INITIAL clause as applicable. In all other cases, the contents of the data items referenced by data-name-1 through data-name-4 of the CD associated with the INITIAL clause are initialized to spaces.

   The symbolic names are inserted, or the initialization to spaces is completed, prior to the execution of the first Procedure Division statement.

   The execution of a subsequent RECEIVE statement naming the same contents of the data items referenced by data-name-1 through data-name-4 will return the actual message that caused the program to be scheduled. Only at that time will the remainder of the CD be updated.

7. If the Message Control System (MCS) attempts to schedule a program lacking an INITIAL clause, the results are undefined.

8. During the execution of a RECEIVE statement, the Message Control System (MCS) provides in the data item referenced by data-name-5 the date on which it recognized that the message was complete, in the form YYMMDD (year, month, day). The contents of the data items referenced by data-name-5 are not updated by the MCS other than as part of the execution of a RECEIVE statement.

9. During the execution of a RECEIVE statement, the Message Control System (MCS) provides in the data item referenced by data-name-6 the time at which it recognized that the message was complete, in the form HHMMSSTT (hours, minutes, seconds, hundredth of a second). The contents of the data items referenced by data-name-6 are not updated by the MCS other than as part of the execution of a RECEIVE statement.

10. During the execution of a RECEIVE statement, the MCS provides, in the data item referenced by data-name-7, the symbolic name of the communications terminal that is the source of the message being transferred. This symbolic name must follow the rules for the formation of system names. However, if the symbolic name of the communication terminal is not known to the MCS, the contents of the data item referenced by data-name-7 will contain spaces.

11. The MCS indicates via the contents of the data item referenced by data-name-8 the number of character positions filled as a result of the execution of the RECEIVE statement. (See the "RECEIVE Statement", Chapter 12).

12. The contents of the data item referenced by data-name-9 are set only by the MCS as part of the execution of a RECEIVE statement according to the following rules:

    a. When the RECEIVE MESSAGE phrase is specified, then:

       (i) If an end of group has been detected, the contents of the data item referenced by data-name-9 are set to 3.

       (ii) If an end of message has been detected, the contents of the data item referenced by data-name-9 are set to 2.

       (iii) If less than a message is transferred, the contents of the data item referenced by data-name-9 are set to 0.

    b. When the RECEIVE SEGMENT phrase is specified, then:

       (i) If an end of group has been detected, the contents of the data item referenced by data-name-9 are set to 3.

       (ii) If an end of message has been detected, the contents of the data item referenced by data-name-9 are set to 2.

       (iii) If an end of segment has been detected, the contents of the data item referenced by data-name-9 are set to 1.

       (iv) If less than a message segment is transferred, the contents of the data item referenced by data-name-9 are set to 0.

    c. When more than one of the above conditions is satisfied simultaneously, the rule first satisfied in the order listed determines the contents of the data-item referenced by data-name-9.

13. The contents of the data item referenced by data-name-10 indicate the status condition of the previously executed RECEIVE, ACCEPT MESSAGE COUNT, ENABLE INPUT, or DISABLE INPUT statement.

    The actual association between the contents of the data item referenced by data-name-10 and the status condition itself is defined in the table "Communication Status Key Condition" below.

14. The contents of the data item referenced by data-name-11 indicate the number of messages that exist in a queue, sub-queue-1, ... The MCS updates the contents of the data item referenced by data-name-11 only as part of the execution of an ACCEPT MESSAGE COUNT statement.

15. During the execution of a RECEIVE, an ENABLE INPUT, or a DISABLE INPUT statement, the queue specified in the CD must have been defined as an input queue; |otherwise, the statement will not be executed and the STATUS KEY item will be updated with the code 99 (see "Communication Status Key Condition Table", below).|

Format 2

16. The nature of the output CD information is such that it is not sent to the terminal, but constitutes the communication between the program and the MCS about the message being handled.

17. For each output CD entry, a record area of contiguous character positions is allocated according to the following formula: (10 plus (13 times integer-1)). The implicit description of this record area is:

a. The DESTINATION COUNT clause defines data-name-1 as the name of a data item whose implicit description is that of an integer of 4 digits, without an operational sign, occupying character positions 1 through 4 in the record.

b. The TEXT LENGTH clause defines data-name-2 as the name of an elementary data item whose implicit description is that of an integer of 4 digits, without an operational sign, occupying character positions 5 through 8 in the record.

c. The STATUS KEY clause defines data-name-3 to be an elementary alphanumeric data item of 2 characters occupying positions 9 and 10 in the record.

d. Character positions 11 through 23 and every set of 13 characters thereafter will form table items of the following description:

   (i) The ERROR KEY clause defines data-name-4 as the name of an elementary alphanumeric data item of 1 character.

   (ii) The SYMBOLIC DESTINATION clause defines data-name-5 as the name of an elementary alphanumeric data item of 12 characters.

Use of the above clauses results in a record whose implicit description is equivalent to the following:

```
IMPLICIT DESCRIPTION       COMMENT

01 data-name-0.
   02 data-name-1         PICTURE 9(04).      DESTINATION COUNT
   02 data-name-2         PICTURE 9(04).      TEXT LENGTH
   02 data-name-3         PICTURE XX.         STATUS KEY
   02 data-name OCCURS    DESTINATION TABLE
   integer-1 TIMES.
      03 data-name-4       PICTURE X.          ERROR KEY
      03 data-name-5       PICTURE X(12).      SYMBOLIC DESTINATION
```

In the above, the information given under 'COMMENT' is for clarification and is not part of the Data Description.

18. During the execution of a SEND, PURGE, ENABLE OUTPUT or DISABLE OUTPUT statement, the contents of the data item referenced by data-name-1 will indicate to the MCS the number of symbolic destinations that are to be used from the area referenced by data-name-5.

    The MCS finds the first symbolic destination name in the first occurrence of the area referenced by data-name-5, the second symbolic destination name in the second occurrence of the area referenced by data-name-5, ... , up to and including the occurrence of the area referenced by data-name-5 indicated by the contents of data-name-1.

    If during the execution of a SEND, PURGE, ENABLE OUTPUT or DISABLE OUTPUT statement the value of the data item referenced by data-name-1 is not in the range of 1 through integer-1, an error condition is indicated, no action is taken, and the execution of the SEND, PURGE, ENABLE OUTPUT or DISABLE OUTPUT statement is terminated.

19. It is the responsibility of the user to insure that the values of the data items referenced by data-name-1, data-name-2 and data-name-5 are valid at the time of execution of the SEND, PURGE, ENABLE OUTPUT, or DISABLE OUTPUT statement.

20. As part of the execution of a SEND statement, the MCS will interpret the contents of the data item referenced by data-name-2 to be the user's indication of the number of leftmost character positions of the data item referenced by the identifier, in the associated SEND statement, from which the data is to be transferred (see the "SEND Statement", Chapter 13).

21. Each occurrence of the data item referenced by data-name-5 contains a symbolic destination name previously known to the MCS. These symbolic destination names must follow the rules for the formation of system-names.

22. The contents of the data item referenced by data-name-3 indicate the status condition of the previously executed SEND, PURGE, ENABLE OUTPUT, or DISABLE OUTPUT statement.

    The actual association between the contents of the data item referenced by data-name-3 and the status condition itself is defined in the table "Communication Status Key Condition" below.

23. If, during the execution of a DISABLE OUTPUT, ENABLE OUTPUT, PURGE or SEND statement, the MCS determines an error has occurred, the contents of the data item referenced by data-name-3 and the contents of each occurrence of data-name-4, up to and including the occurrence specified by the contents of data-name-1 are updated.

    The actual association between the contents of the data item referenced by data-name-4 and the error condition itself is defined in the table "Error Key Values" below.

24. During the execution of a SEND, an ENABLE OUTPUT, or a DISABLE OUTPUT statement, the queue specified in the CD must have been defined as an output queue or the queue must correspond to a terminal which is already logged on with the application; |otherwise, the statement will not be executed and the STATUS KEY item will be updated with the code 9F.|

Format 3

25. The input-output CD information constitutes the communication between the MCS and the program about the message being handled. This information does not come from the terminal as part of the message.

26. For each input-output CD, a record area of 33 contiguous character positions is allocated. This record area is defined to the MCS as follows:

    a. The MESSAGE DATE clause defines data-name-1 as the name of a data item whose implicit description is that of an integer of 6 digits, without an operational sign, occupying character positions 1 through 6 in the record.

    b. The MESSAGE TIME clause defines data-name-2 as the name of a data item whose implicit description is that of an integer of 8 digits, without an operational sign, occupying character positions 7 through 14 in the record.

    c. The SYMBOLIC TERMINAL clause defines data-name-3 as the name of an elementary alphanumeric data item of 12 characters occupying positions 15 through 26 in the record.

    d. The TEXT LENGTH clause defines data-name-4 as the name of an elementary data item whose implicit description is that of an integer of 4 digits, without an operational sign, occupying character positions 27 through 30 in the record.

    e. The END KEY clause defines data-name-5 as the name of an elementary alphanumeric data item of 1 character occupying position 31 in the record.

    f. The STATUS KEY clause defines data-name-6 as the name of an elementary data item of 2 characters occupying positions 32 and 33 in the record.

    The second option may be used to replace the above clauses by a series of data-names which, taken in order, correspond to the data-names defined by these clause.

Use of either option results in a record whose implicit description is equivalent to the following:

IMPLICIT DESCRIPTION	COMMENT	
01 data-name-0.	PICTURE 9(06).	MESSAGE DATE
02 data-name-1		
02 data-name-2	PICTURE 9(08).	MESSAGE TIME
02 data-name-3	PICTURE X(12).	SYMBOLIC TERMINAL
02 data-name-4	PICTURE 9(04).	TEXT LENGTH
02 data-name-5	PICTURE X.	END KEY
02 data-name-6	PICTURE XX.	STATUS KEY

In the above, the information under 'COMMENT' is for clarification and is not part of the data description.

27. When a program is scheduled by the MCS to process a message, the first RECEIVE statement referencing the input-output CD with the INITIAL clause returns the actual message that caused the program to be scheduled.

28. Data-name-1 has the format (YYMMDD) (year, month, day). Its contents represent the date on which the MCS recognizes that the message is complete.

    The contents of the data item referenced by data-name-1 are updated only by the MCS as part of a RECEIVE statement.

29. Data-name-2 has the format 'HHMMSSTT' (hours, minutes, seconds, hundredth of a second) and its contents represent the time at which the MCS recognizes that the message is complete.

    The contents of the data referenced by data-name-2 are updated only by the MCS as part of the execution of a RECEIVE statement.

30. Whenever a program is scheduled by the MCS to process a message, that program establishes a run unit and the symbolic name of the communication terminal that is the source of the message that invoked this program is placed in the data item referenced by data-name-3 of the input-output CD associated with the INITIAL clause as applicable. This symbolic name must follow the rules for the formation of system-names.

    In all other cases, the contents of the data item referenced by data-name-3 of the input-output CD associated with the INITIAL clause are initialized to spaces.

    The symbolic name is inserted, or the initialization to spaces is completed, prior to the execution of the first Procedure Division statement.

31. If the Message Control System (MCS) attempts to schedule a program lacking an INITIAL clause, the results are undefined.

32. When the INITIAL clause is specified for an input-output CD and the program is scheduled by the MCS, the contents of the data item referenced by data-name-3 must not be changed by the program. If these contents are changed, the execution of any statement referencing cd-name is unsuccessful, and the data item referenced by data-name-6 is set to indicate unknown source or destination, as applicable (see the table "Communication Status Key Condition" below).

33. For an input-output CD without the INITIAL clause, or for an input-output CD with the INITIAL clause when the program is not scheduled by the MCS, the program must specify the symbolic name of the source or destination in data-name-3 prior to the execution of the first statement referencing cd-name.

    After executing the first statement referencing cd-name, the contents of the data item referenced by data-name-3 must not be changed by the program. If these contents are changed, the execution of any statement referencing cd-name is unsuccessful, and the data item referenced by data-name-6 is set to indicate unknown source or destination, as applicable (see the table "Communication Status Key Condition" below).

34. The Message Control System indicates, through the contents of the data item referenced by data-name-4, the number of character positions filled as a result of the execution of the RECEIVE statement (see the "RECEIVE Statement", Chapter 12).

    As part of the execution of a SEND statement, the MCS interprets the contents of the data item referenced by data-name-4 as the user's indication of the number of the leftmost character positions of the data item referenced by the associated SEND identifier from which data is transferred (see the "SEND Statement", Chapter 13).

35. The contents of the data item referenced by data-name-5 are set only by the MCS as part of the execution of a RECEIVE statement according to the following rules:

    a. When the RECEIVE MESSAGE phrase is specified, then:

       (i) If an end of group has been detected, the contents of the data item referenced by data-name-5 are set to 3.

       (ii) If an end of message has been detected, the contents of the data item referenced by data-name-5 are set to 2.

       (iii) If less than a message is transferred, the contents of the data item referenced by data-name-5 are set to 0.

    b. When the RECEIVE SEGMENT phrase is specified, then:

       (i) If an end of group has been detected, the contents of the data item referenced by data-name-5 are set to 3.

       (ii) If an end of message has been detected, the contents of the data item referenced by data-name-5 are set to 2.

       (iii) If an end of segment has been detected, the contents of the data item referenced by data-name-5 are set to 1.

       (iv) If less than a message is transferred, the contents of the data item referenced by data-name-5 are set to 0.

    c. When more than one of the above conditions is satisfied simultaneously, the rule first satisfied in the order listed determines the contents of the data item referenced by data-name-5.

36. The contents of the data item referenced by data-name-6 indicates the status condition of the previously executed DISABLE, ENABLE, PURGE, RECEIVE or SEND statement.

    The actual association between the contents of the data item referenced by data-name-6 and the status condition itself is defined in the table "Communication Status Key Condition" below.

All Formats

The following table indicates the possible contents of the data items referenced by data-name-10 for Format 1, by data-name-3 for Format 2 and by data-name-6 for format 3 at the completion of each statement shown. An "X" on a line in a statement column indicates that the associated code shown for that line is possible for that statement.

|Status codes from '91' through '9G' are DPS 7 extensions to the American National Standard.|

Table 8-1. Communication Status Key Condition (1/2)

STATUS KEY CODE	ACCEPT (with COUNT)	DISABLE input (with TERMINAL)	DISABLE input (without TERMINAL)	DISABLE output	ENABLE input (with TERMINAL)	ENABLE input (without TERMINAL)	ENABLE output	PURGE	RECEIVE	SEND input-output	SEND output	
00	X	X	X	X	X	X	X	X		X	X	No error detected, action completed.
10								X			X	1 or more destinations disabled, action completed.
15		X	X	X	X	X	X					Symbolic source or 1 or more queues or destinations already disabled/enabled.
20	X		X			X			X			1 or more queues/subqueues unknown*, no action taken.
20		X			X							Source unknown*, no action taken.
20				X			X	X		X	X	No action taken for 1 or more destinations unknown*. Action taken for known destinations. data-name-4 (ERROR KEY) indicates known or unknown
21		X			X				X			Symbolic source is unknown. No action taken.
30				X			X	X			X	DESTINATION COUNT invalid, no action taken
40		X	X	X	X	X	X					Password invalid, no enabling/disabling action taken.
50									X	X		Character count > length of sending field, no action taken.
60									X	X		Partial segment with 0 character count or no sending area specified, no action taken.
65											X	Output queue capacity exceeded.
70							X					1 or more destinations do not have portions associated with them. Action completed for other destinations.
80			X	X		X	X	X			X	A combination of at least two status key conditions 10, 15 and 20 have occurred.
91										X	X	Message data not transferred to queue due to unavailability of mass storage.
92									X	X	X	Message data not transferred due to unavailability of memory space.
93									X			No data can be input from the terminal to the queue to which a DISABLE statement has been issued.
94										X	X	All message data not transferred because maximum message size exceeded, message truncated.
95									X			Message too long. Truncated to maximum size specified.
95										X	X	Message discarded due to queue allocation overflow.

Table 8-1. Communication Status Key Condition (2/2)

STATUS KEY CODE	ACCEPT (with COUNT)	DISABLE input (with TERMINAL)	DISABLE input (without TERMINAL)	DISABLE output	ENABLE input (with TERMINAL)	ENABLE input (without TERMINAL)	ENABLE output	PURGE	RECEIVE	SEND input-output	SEND output	
96									X			Message data returned but at least 1 previous message lost.
97										X	X	identifier-2 (see SEND statement) differ "0", "1", "2" or "3"
98									X	X	X	Message data not transferred due to I/O error on disk file.
99		X	X		X	X			X			Access to queue in conflict with JCL definition.
9A									X			BREAK has been detected, queue corresponding to symbolic source has been disabled.
9B									X			RVI has been detected, queue corresponding to symbolic source has been disabled.
9C									X			Terminal corresponding to symbolic source has been disconnected.
9D									X			Terminal corresponding to symbolic source has been disconnected.
9E									X			Shutdown is announced, application is required to terminate.
9F				X			X			X	X	Access to queue in conflict with JCL definition, or related terminal not logged on to application.
9G										X	X	Message not transferred, checkpoint should be taken before attempting further data transfers. Applicable to queues with the restart option (controlled or ROLLBACK).

* unknown means symbolic queue not defined in JCL.

NOTE: Status codes from '9A' through '9G' are available to an application only if the related program queue has been defined with the 'BREAK' option in the Communication Network Configuration (See the manual *GCOS 7-V6 Networks: Overview and Generation*).

The table below indicates the possible contents of the data item referenced by data-name-4 for format 2 at the completion of each statement shown. An 'X' on a line in a statement column indicates that the associated error key value shown for that line is possible for that statement.

Table 8-2. Error Key Values

SEND	PURGE	ENABLE OUTPUT	DISABLE OUTPUT	ERROR KEY VALUE	
X	X	X	X	0	No error.
X	X	X	X	1	Symbolic destination unknown.
X	X			2	Symbolic destination disabled.
	X			4	No partial message with referenced symbolic destination.
		X	X	5	Symbolic destination already enabled/disabled.
X				6	Output queue capacity exceeded.
				7-9	Reserved for future use.

8.11 REPORT DESCRIPTION - COMPLETE ENTRY SKELETON

Description

The Report Description entry names a report, specifies any identifying characters to be prefixed to each print line, and describes the physical structure and organization of that report.

Format

```
RD report-name [IS GLOBAL]

      [CODE literal]

      {CONTROL IS  } {{data-name-1}...        }
     [{            } {                        }]
      {CONTROLS ARE} { FINAL [data-name-1]... }

            [LIMIT IS  ]           [LINE ]
      [PAGE [          ] integer-1 [     ] [HEADING integer-2]
            [LIMITS ARE]           [LINES]

           [FIRST DETAIL integer-3] [LAST DETAIL integer-4]

           [FOOTING integer-5]].
```

Syntax Rules

1. The report-name must appear in one and only one REPORT clause.

2. The order of appearance of the clauses following the report-name is immaterial.

3. Report-name is the highest permissible qualifier that may be specified for LINE-COUNTER, PAGE-COUNTER and all data-names defined within the Report Section.

PAGE-COUNTER Rules

1. PAGE-COUNTER is the reserved word used to reference a special register that is automatically created for each report specified in the Report Section (See "Special Registers", Chapter 3).

2. In the Report Section, a reference to PAGE-COUNTER can only appear in a SOURCE clause. In the Procedure Division, PAGE-COUNTER may be used in any context in which a data item with an integral value can appear.

3. If more than one PAGE-COUNTER exists in a program, PAGE-COUNTER must be qualified by a report-name whenever it is referenced in the Procedure Division.

   In the Report Section an unqualified reference to PAGE-COUNTER is qualified implicitly by the name of the report in whose Report Description entry the reference is made. Whenever the PAGE-COUNTER of a different report is referenced, PAGE-COUNTER must be explicitly qualified by the report-name associated with the different report.

4. Execution of the INITIATE statement causes the Report Writer Control System to set the PAGE-COUNTER of the referenced report to one (1).

5. PAGE-COUNTER is automatically incremented by one (1) each time the Report Writer Control System executes a page advance.

6. PAGE-COUNTER may be altered by Procedure Division statements.

LINE-COUNTER Rules

1. LINE-COUNTER is the reserved word used to reference a special register that is automatically created for each report specified in the Report Section (See "Special Registers", Chapter 3).

2. In the Report Section a reference to LINE-COUNTER can only appear in a SOURCE clause. In the Procedure Division, LINE-COUNTER may be used in any context in which a data item with an integral value may appear. However, only the Report Writer Control System can change the contents of LINE-COUNTER.

3. If more than one LINE-COUNTER exists in a program, LINE-COUNTER must be qualified by a report-name whenever it is referenced in the Procedure Division.

   In the Report Section an unqualified reference to LINE-COUNTER is qualified implicitly by the name of the report in whose Report Description entry the reference is made. Whenever the LINE-COUNTER of a different report is referenced, LINE-COUNTER must be explicitly qualified by that report-name associated with the different report.

4. Execution of an INITIATE statement causes the Report Writer Control System to set the LINE-COUNTER of the referenced report to zero (0). The Report Writer Control System also automatically resets LINE-COUNTER to zero each time it executes a page advance.

5. The value of LINE-COUNTER is not affected by the processing of non-printable report groups nor by the processing of a printable report group whose printing is suppressed by means of the SUPPRESS statement.

6. At the time each print line is presented, the value of LINE-COUNTER represents the line number on which the print line is presented. The value of LINE-COUNTER after the presentation of a report group is governed by the presentation rules for the report group (See "Presentation Rules Tables", this chapter).

8.12 DATA DESCRIPTION - COMPLETE ENTRY SKELETON

Description

A Data Description entry specifies the characteristics of a particular item of data.

Format 1

```
             [data-name]
level-number [         ]
             [FILLER   ]

    [REDEFINES data-name]

    [IS EXTERNAL]

    [IS GLOBAL]

    {PICTURE}                       |--------------------------|
   [{       } IS character-string   | [DEPENDING ON data-name] |
    {PIC    }                      |--------------------------|

                 {   BINARY             }
                 { |-----|              }
                 { | BIT |              }
                 { |-----|              }
                 {   COMPUTATIONAL      }
                 {   COMP               }
                 { |------------------| }
                 { | COMPUTATIONAL-1  | }
                 { | COMP-1           | }
                 { | COMPUTATIONAL-2  | }
                 { | COMP-2           | }
                 { | COMPUTATIONAL-3  | }
                 { | COMP-3           | }
   [[USAGE IS]   { | COMPUTATIONAL-5  | }
                 { | COMP-5           | }
                 { | COMPUTATIONAL-8  | }
                 { | COMP-8           | }
                 { | COMPUTATIONAL-9  | }
                 { | COMP-9           | }
                 { | COMPUTATIONAL-10 | }
                 { | COMP-10          | }
                 { | COMPUTATIONAL-15 | }
                 { | COMP-15          | }
                 { | POINTER          | }
                 { |------------------| }
                 {   DISPLAY            }
                 {   INDEX              }
                 {   PACKED-DECIMAL     }
```

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

 [OCCURS integer TIMES                                          ]
 [                                                              ]
 [    {ASCENDING }                                             ]
 [   [{          } KEY IS {data-name-4}... ]...                ]
 [    {DESCENDING}                                             ]
 [                                                              ]
 [    [INDEXED BY {index-name-1}... ]                           ]
 [                                                              ]
 [OCCURS integer-2 TO integer-1 TIMES DEPENDING ON data-name]
 [                                                              ]
 [    {ASCENDING }                                             ]
 [   [{          } KEY IS {data-name-4}... ]...                ]
 [    {DESCENDING}                                             ]
 [                                                              ]
 [    [INDEXED BY {index-name-1}... ]                           ]

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

 {JUSTIFIED}
[{         } RIGHT]
 {JUST     }

 [BLANK WHEN ZERO]

           { literal-1}
 [VALUE IS { |------| }] .
           { | NULL | }
             |------|
```

Format 2

```
66 data-name-1
                          {THROUGH}
     RENAMES data-name-2 [{       } data-name-3].
                          {THRU   }
```

Format 3

```
88 condition-name
                  |------|
                { | NULL |                                 }
    {VALUE IS  } { |------|                                 }
    {          } {                {THROUGH}                 }
    {VALUES ARE} {  {literal-1 [{        } literal]-2}... }
                 {                {THRU   }                 }

      |----------------------------------|
      | [WHEN SET TO FALSE IS literal-3] | .
      |----------------------------------|
```

Syntax Rules

1. Level-number in Format 1 may be any number from 01 through 49, or 77 (See the "Level-Number Clause", Chapter 9).

2. In Format 1, the data-name-1 or FILLER clause, if specified, must immediately follow the level-number; the REDEFINES clause, if specified, must immediately follow the data-name-1 or FILLER clause, if either is specified; otherwise, it must immediately follow the level-number. The remaining clauses may be written in any order.

3. The EXTERNAL clause may be specified only in Data Description entries whose level-numbers are 01 |or 77, in the Working-Storage Section and the Constant Section.|

4. The EXTERNAL clause and the REDEFINES clause must not be specified in the same Data Description entry.

5. The GLOBAL clause may be specified only in Data Description entries whose level-number is 01.

6. Data-name-1 in format 1 must be specified for any entry containing the GLOBAL or EXTERNAL clause or for record descriptions associated with a File Description entry which contains the EXTERNAL or GLOBAL clause.

7. The PICTURE clause must be specified for every elementary item except |a COMP-9, COMP-10, COMP-15, POINTER item or | an index data item, and the subject of the RENAMES clause, in which case use of this clause is prohibited, |or a COMP-1 or COMP-2 item, in which case use of this clause is optional.

8. The words THRU and THROUGH are equivalent.

General Rules

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

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

   a. Another condition-name
   b. A level 66 item
   c. A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED or USAGE (other than USAGE IS DISPLAY)
   d. An index data item

3. Multiple level 01 entries subordinate to any given level indicator other than the level indicator RD for Report Description entries, represent implicit re-definitions of the same area.

8.13 REPORT GROUP DESCRIPTION - COMPLETE ENTRY SKELETON

Description

The Report Group Description entry specifies the characteristics of a report group and of the individual items within a report group.

Format 1

```
01 [data-name-1]2
                    {integer-1 [ON NEXT PAGE]}
    [LINE NUMBER IS {                        }]
                    {PLUS integer-2          }

                   {integer-3     }
    [NEXT GROUP IS {PLUS integer-4}]
                   {NEXT PAGE     }

             {{REPORT HEADING}               }
             {{RH            }               }
             {                               }
             {{PAGE HEADING}                 }
             {{PH          }                 }
             {                               }
             {{CONTROL HEADING} {data-name-2}}
             {{CH              } {FINAL      }}
             {                               }
      TYPE IS {{DETAIL}                      }
             {{DE    }                       }
             {                               }
             {{CONTROL FOOTING} {data-name-3}}
             {{CF              } {FINAL      }}
             {                               }
             {{PAGE FOOTING}                 }
             {{PF          }                 }
             {                               }
             {{REPORT FOOTING}               }
             {{RF            }               }

   [[USAGE IS] DISPLAY].
```

Format 2

```
level-number [data-name-1]

                    {integer-1 [ON NEXT PAGE]}
    [LINE NUMBER IS {                        }]
                    {PLUS integer-2          }

   [[USAGE IS] DISPLAY].
```

Format 3

```
level-number [data-name-1]

    {PICTURE}
    {       } IS character-string
    {PIC    }

   [[USAGE IS] DISPLAY]

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

    {JUSTIFIED}
   [{         } RIGHT]
    {JUST     }

    [BLANK WHEN ZERO]

                    {integer-1 [ON NEXT PAGE]}
    [LINE NUMBER IS {                        }]
                    {PLUS integer-2          }

    [COLUMN NUMBER IS integer-3]

   { SOURCE IS identifier-1                              }
   {                                                     }
   { VALUE IS literal-1                                  }
   {                                                     }
   {{SUM {identifier-2}... [UPON {data-name-2}... ]}...}
   {                                                     }
   {                 {data-name-3}                       }
   {     [RESET ON {            }]                       }
                     {FINAL      }

    [GROUP INDICATE].
```

Syntax Rules

1. The Report Group Description entry can appear only in the Report Section.

2. Except for the data-name clause, which when present must immediately follow the level-number, the clauses may be written in any sequence.

3. In Format 2 the level-number may be any integer from 02 to 48 inclusive. In Format 3 the level-number may be any integer from 02 to 49 inclusive.

4. A description of a report group may consist of one, two or three hierarchic levels:

   a. The first entry that describes a report group must be a Format 1 entry.

   b. Both Format 2 and Format 3 entries may be immediately subordinate to a Format 1 entry.

   c. At least one Format 3 entry must be immediately subordinate to a Format 2 entry.

   d. Format 3 entries must define elementary data items.

5. In a Format 1 entry, data-name-1 is required only when:

   a. A DETAIL report group is referenced by a GENERATE statement,

   b. A DETAIL report group is referenced by the UPON phrase of a SUM clause,

   c. A report group is referenced in a USE BEFORE REPORTING sentence,

   d. The name of a CONTROL FOOTING report group is used to qualify a reference to a sum counter.

   If specified, data-name-1 may be referenced only by a GENERATE statement, the UPON phrase of a SUM clause, a USE BEFORE REPORTING sentence, or as a sum counter qualifier.

6. A Format 2 entry must contain at least one optional clause.

7. In a Format 2 entry, data-name-1 is optional. If present it may be used only to qualify a sum counter reference.

8. In the Report Section, the USAGE clause is used only to declare the usage of printable items.

   a. If the USAGE clause appears in a Format 3 entry, that entry must define a printable item.

   b. If the USAGE clause appears in a Format 1 or Format 2 entry, at least one subordinate entry must define a printable item.

9. An entry that contains a LINE NUMBER clause must not have a subordinate entry that also contains a LINE NUMBER clause.

10. In Format 3:

    a. A GROUP INDICATE clause may appear only in a TYPE DETAIL report group.

    b. A SUM clause may appear only in a TYPE CONTROL FOOTING report group.

    c. An entry that contains a COLUMN NUMBER clause but no LINE NUMBER clause must be subordinate to an entry that contains a LINE NUMBER clause.

    d. Data-name-1 is optional but may be specified in any entry. Data-name-1 may be referenced only if the entry defines a sum counter.

    e. An entry that contains a VALUE clause must also have a COLUMN NUMBER clause.

11. The following table shows all permissible clause combinations for a Format 3 entry. The table is read from left to right along the selected row.

    An 'M' indicates that the presence of the clause is mandatory.

    A 'P' indicates that the presence of the clause is permitted, but not required.

    A blank indicates that the clause is not permitted.

Table 8-3. Permissible Clause Combinations in Format 3 Entries

PIC	COLUMN	SOURCE	SUM	VALUE	JUST	BLANK WHEN ZERO	GROUP INDICATE	USAGE	LINE	SIGN
M			M						P	P
M	M		M			P		P	P	P
M	P	M			P		P	P	P	P
M	P	M				P	P	P	P	P
M	M			M	P		P	P	P	P

General Rules

Format 1 is the report group entry. The report group is defined by the contents of this entry and all of its subordinate entries.

8.13.1 Presentation Rules Tables

Description

The tables and rules, below, specify:

1. The permissible combinations of LINE NUMBER and NEXT GROUP clauses for each type of report group,

2. The requirements that are placed on the use of these clauses, and

3. The interpretation that the RWCS gives to these clauses.

Organization

There is an individual Presentation Rules Table for each of the following types of report groups: REPORT HEADING, PAGE HEADING, PAGE FOOTING, REPORT FOOTING. In addition, DETAIL report groups, CONTROL HEADING report groups, and CONTROL FOOTING report groups are treated jointly in the Body Group Presentation Rules Table (See the "Body Group Presentation Rules", this chapter).

Columns 1 and 2 of a Presentation Rules Table list all of the permissible combinations of LINE NUMBER and NEXT GROUP clauses for the designated report group TYPE. Consequently, for the purpose of identifying the set of presentation rules that apply to a particular combination of LINE NUMBER and NEXT GROUP clauses, a Presentation Rules Table is read from left to right, along the selected row.

The Applicable Rules columns of a Presentation Rules Table are partitioned into two parts. The first part specifies the rules that apply if the report description contains a PAGE clause, and the second part specifies the rules that apply if the PAGE clause is omitted. The purpose of the rules named in the Applicable Rules columns is discussed below:

1. Upper Limit Rules and Lower Limit Rules

   These rules specify the vertical subdivisions of the page within which the specified report group may be presented.

   In the absence of a PAGE clause the printed report is not considered to be partitioned into vertical subdivisions. Consequently, within the Tables no Upper Limit Rule and Lower Limit Rule is specified for a report description in which the PAGE clause is omitted.

2. Fit Test Rules

   The Fit Test Rules are applicable only to body groups, and hence Fit Test Rules are specified only within the Body Group Presentation Rules Table. At object time the RWCS applies the Fit Test Rules to determine whether the designated body group can be presented on the page to which the report is currently positioned.

   However, even for body groups there are no Fit Test Rules when the PAGE clause is omitted from the Report Description entry.

3. First Print Line Position Rules

   The First Print Line Position Rules specify where on the report medium the RWCS shall present the first print line of the given report group.

   The Presentation Rules Tables do not specify where on the report medium the RWCS shall present the second and subsequent print lines (if any) of a report group. Certain general rules determine where the second and subsequent print lines of a report group shall be presented. For this information, refer to the "General Rules" of the "LINE NUMBER Clause" in Chapter 9.

4. Next Group Rules

   The next Group Rules relate to the proper use of the NEXT GROUP clause.

5. Final LINE-COUNTER Setting Rules

   The terminal values that the RWCS places in LINE-COUNTER after presenting report groups are specified by the Final LINE-COUNTER Setting Rules.

LINE NUMBER Clause Notation

Column 1 of the Presentation Rules Table uses a shorthand notation to describe the sequence of LINE NUMBER clauses that may appear in the description of a report group. The meaning of the abbreviations used in column 1 is as follows:

1. The letter 'A' represents one or more absolute LINE NUMBER clauses, none of which have the NEXT PAGE phrase, that appear in consecutive order within the sequence of LINE NUMBER clauses in the Report Group Description entry.

2. The letter 'R' represents one or more relative LINE NUMBER clauses that appear in consecutive order within the sequence of LINE NUMBER clauses in the Report Group Description entry.

3. The letters 'NP' represent one or more absolute LINE NUMBER clauses that appear in consecutive order within the sequence of LINE NUMBER clauses in the Report Group Description entry with the phrase NEXT PAGE appearing in the first and only in the first LINE NUMBER clause.

   When two abbreviations appear together, they refer to a sequence of LINE NUMBER clauses that consist of two specified consecutive sequences. For example 'AR' refers to a Report Group Description entry within which the 'A' sequence (defined in rule 1 above) is immediately followed by the 'R' sequence (defined in rule 2 above).

LINE NUMBER Clause Sequence Substitutions

Where 'AR' is shown to be a permissible sequence in the Presentation Rules Table, 'A' is also permissible and the same presentation rules are applicable.

Where 'NP R' is shown to be a permissible sequence in the Presentation Rules Table, 'NP' is also permissible and the same presentation rules are applicable.

Saved Next Group Integer Description

Saved Next Group Integer is a data item that is addressable only by the RWCS. When an absolute NEXT GROUP clause specifies a vertical positioning value which cannot be accommodated on the current page, the RWCS stores that value in Saved Next Group Integer. After page advance processing, the RWCS positions the next body group using the value stored in Saved Next Group Integer.

8.13.2 REPORT HEADING Group Presentation Rules

*The following table points to the appropriate Presentation Rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a REPORT HEADING report group.

Table 8-4. REPORT HEADING Group Presentation Rules

		APPLICABLE RULES ***						
**		If the PAGE clause is specified					If the PAGE clause is omitted	
Sequence of LINE NUMBER clauses *	NEXT GROUP clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final Line-Counter Setting	First Print Line Position	Final Line Counter Setting
A R	Absolute	1	2a	3a	4a	5a	Illegal Combination (see the LINE NUMBER clause)	
AR	Relative	1	2a	3a	4b	5b	Illegal Combination (see the LINE NUMBER clause)	
AR	NEXT PAGE	1	2b	3a	4c	5c	Illegal Combination (see the LINE NUMBER clause)	
AR		1	2a	3a		5d	Illegal Combination (see the LINE NUMBER clause)	
R	Absolute	1	2a	3b	4a	5a	Illegal Combination (see the LINE NUMBER clause)	
R	Relative	1	2a	3b	4b	5b	3d	5b
R	NEXT PAGE	1	2b	3b	4c	5c	Illegal Combination (see the LINE NUMBER clause)	
R		1	2a	3b		5d	3d	5d
				3c		5e	3c	5e

***** The meaning of the abbreviations used in column 1 has been previously stated. (See "LINE NUMBER Clause Notation", above).

****** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the Report Group Description entry.

******* A blank entry in the Applicable Rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

1. Upper Limit Rule

   The First line number on which the REPORT HEADING report group can be presented is the line number specified by the HEADING phrase of the PAGE clause.

2. Lower Limit Rules

   a. The last line number on which the REPORT HEADING report group can be presented is the line number that is obtained by subtracting 1 from the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.

   b. The last line number on which the REPORT HEADING report group can be presented is the line number specified by integer-1 of the PAGE clause.

3. First Print Line Position Rules

   a. The first print line of the REPORT HEADING report group is presented on the line number specified by the integer of its LINE NUMBER clause.

   b. The first print line of the REPORT HEADING report group is presented on the line number obtained by adding the integer of the first LINE NUMBER clause and the value obtained by subtracting 1 from the value of integer-2 of the HEADING phrase of the PAGE clause.

   c. The REPORT HEADING report group is not presented.

   d. The first print line of the REPORT HEADING report group is presented on the line number obtained by adding the contents of its LINE-COUNTER (in this case, zero) to the integer of the first LINE NUMBER clause.

4. Next Group Rules

   a. The NEXT GROUP integer must be greater than the line number on which the final print line of the REPORT HEADING report group is presented. In addition, the NEXT GROUP integer must be less than the line number specified by the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.

   b. The sum of the NEXT GROUP integer and the line number on which the final print line of the REPORT HEADING report group is presented must be less than the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.

   c. NEXT GROUP NEXT PAGE signifies that the REPORT HEADING report group is to be presented entirely by itself on the first page of the report. The RWCS processes no other report group while positioned to the first page of the report.

5. Final LINE-COUNTER Setting Rules

   a. After the REPORT HEADING report group is presented, The RWCS places the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting.

   b. After the REPORT HEADING report group is presented, the RWCS places the sum of the NEXT GROUP integer and the line number on which the final print line of the REPORT HEADING report group was presented into LINE-COUNTER as the final LINE-COUNTER setting.

   c. After the REPORT HEADING report group is presented, the RWCS places zero into LINE-COUNTER as the final LINE-COUNTER setting.

   d. After the REPORT HEADING report group is presented, the final LINE-COUNTER setting is the line number on which the final print line of the REPORT HEADING report group was presented.

   e. LINE-COUNTER is unaffected by the processing of a non-printable report group.

8.13.3 PAGE HEADING Group Presentation Rules

The table below points to the appropriate Presentation Rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a PAGE HEADING report group.

Table 8-5. PAGE HEADING Group Presentation Rules

		APPLICABLE RULES ***				
******		**If the PAGE clause is specified ******				
Sequence of LINE NUMBER clauses *	NEXT GROUP clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final Line-Counter Setting
A R		1	2	3a		4a
R		1	2	3b		4a
				3c		4b

***** The meaning of the abbreviations used in column 1 has been previously stated. (See "LINE NUMBER Clause Notation" above.)

****** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the Report Group Description entry.

******* If the PAGE clause is omitted from the Report Description entry, then a PAGE HEADING report group may not be defined. (See the "TYPE Clause", Chapter 9).

******** A blank entry in an Applicable Rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

Presentation Rules

1. Upper Limit Rule

   If a REPORT HEADING report group has been presented on the page on which the PAGE HEADING report group is to be presented, then the first line number on which the PAGE HEADING report group can be presented is one greater than the final LINE-COUNTER setting established by the REPORT HEADING. Otherwise the first line number on which the PAGE HEADING report group can be presented is the line number specified by the HEADING phrase of the PAGE clause.

2. Lower Limit Rule

   The last line number on which the PAGE HEADING report group can be presented is the line number that is obtained by subtracting one (1) from the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.

3. First Print Line Position Rules

   a. The first print line of the PAGE HEADING report group is presented on the line number specified by the integer of its LINE NUMBER clause.

   b. If a REPORT HEADING report group has been presented on the page on which the PAGE HEADING report group is to be presented, then the sum of the final LINE-COUNTER setting established by the REPORT HEADING report group and the integer of the first LINE NUMBER clause of the PAGE HEADING report group defines the line number on which the first print line of the PAGE HEADING report group is presented.

      Otherwise the sum of the integer of the first LINE NUMBER clause of the PAGE HEADING report group and the value obtained by subtracting one (1) from the value of integer-2 of the HEADING phrase of the PAGE clause defines the line number on which the first print line of the PAGE HEADING report group is presented.

   c. The PAGE HEADING report group is not presented.

4. Final LINE-COUNTER Setting Rules

   a. The final LINE-COUNTER setting is the line number on which the final print line of the PAGE HEADING report group was presented.

   b. LINE-COUNTER is unaffected by the processing of a non-printable report group.

8.13.4 Body Group Presentation Rules

The table below points to the appropriate Presentation Rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in CONTROL HEADING, DETAIL and CONTROL FOOTING report groups.

Table 8-6. Body Group Presentation Rules

		APPLICABLE RULES ***							
******		**If the PAGE clause is specified**						**If the PAGE clause is omitted**	
Sequence of LINE NUMBER clauses *	NEXT GROUP clause	Upper Limit	Lower Limit	Fit Test	First Print Line Position	Next Group	Final Line-Counter Setting	First Print Line Position	Final Line Counter Setting
AR	Absolute	1	2a	3a	4a	5	6a	Illegal Combination (see the LINE NUMBER clause)	
AR	Relative	1	2	3a	4a		6b	Illegal Combination (see the LINE NUMBER clause)	
AR	NEXT PAGE	1	2	3a	4a		6c	Illegal Combination (see the LINE NUMBER clause)	
AR		1	2	3a	4a		6d	Illegal Combination (see the LINE NUMBER clause)	
R	Absolute	1	2	3b	4b	5	6a	Illegal Combination (see the LINE NUMBER clause)	
R	Relative	1	2	3b	4b		6b	4d	6f
R	NEXT PAGE	1	2	3b	4b		6c	Illegal Combination (see the LINE NUMBER clause)	
R		1	2	3b	4b		6d	4d	6d
NP R	Absolute	1	2	3c	4a	5	6a	Illegal Combination (see the LINE NUMBER clause)	
NP R	Absolute	1	2	3c	4a		6b	Illegal Combination (see the LINE NUMBER clause)	
NP R	NEXT PAGE	1	2	3c	4a		6c	Illegal Combination (see the LINE NUMBER clause)	

NP R		1	2	3c	4		6d	Illegal Combination (see the LINE NUMBER clause)	
					4c		6e	4c	6e

***** The meaning of the abbreviations used in column 1 has been previously stated. (See "LINE NUMBER Clause Notation", above).

****** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the Report Group Description entry.

******* A blank entry in an Applicable Rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

Presentation Rules

1. Upper Limit Rule

   The first line number on which a body group can be presented is the line number specified by the FIRST DETAIL phrase of the PAGE clause.

2. Lower Limit Rules

   The last line number on which a CONTROL HEADING report group or DETAIL report group can be presented is the line number specified by the LAST DETAIL phrase of the PAGE clause.

   The last line number on which a CONTROL FOOTING report group can be presented is the line number specified by the FOOTING phrase of the PAGE clause.

3. Fit Test Rules

   a. If the value in LINE-COUNTER is less than the integer of the first absolute LINE NUMBER clause, then the body group shall be presented on the page to which the report is currently positioned.

      Otherwise the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS determines whether the Saved Next Group Integer location was set when the final body group was presented on the preceding page. (See final "LINE-COUNTER Setting Rule" 6a.) If Saved Next Group Integer was not so set, the body group shall be presented on the page to which the report is currently positioned. If Saved Next Group Integer was so set, the RWCS moves the Saved Next Group Integer into LINE-COUNTER, resets Saved Next Group Integer to zero, and re-applies Fit Test Rule 3a.

b. If a body group has been presented on the page to which the report is currently positioned, the RWCS computes a trial sum in a work location. The trial sum is computed by adding the contents of LINE-COUNTER to the integers of all LINE NUMBER clauses of the report group. If the trial sum is not greater than the body group's Lower Limit integer, then the report group is presented on the current page. If the trial sum exceeds the body group's Lower Limit integer, then the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS re-applies Fit Test Rule 3b.

   If no body group has yet been presented on the page to which the report is currently positioned, the RWCS determines whether the Saved Next Group Integer location was set when the final body group was presented on the preceding page. (See "Final LINE-COUNTER Setting Rule" 6a, below).

   If Saved Next Group Integer was not so set, the body group shall be presented on the page to which the report is currently positioned.

   If Saved Next Group Integer was so set, the RWCS moves the Saved Next Group Integer into LINE-COUNTER, resets Saved Next Group Integer to zero, and computes a trial sum in a work location.

   The trial sum is computed by adding the contents of LINE-COUNTER to the integer one (1) and the integers of all but the first LINE NUMBER clause of the body group. If the trial sum is not greater than the body group's Lower Limit integer, then the body group is presented on the current page. If the trial sum exceeds the body group's Lower Limit integer, then the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS presents the body group on that page.

c. If a body group has been presented on the page to which the report is currently positioned, the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS re-applies Fit Test Rule 3c.

   If no body group has yet been presented on the page to which the report is currently positioned, the RWCS determines whether the Saved Next Group Integer location was set when the final body group was presented on the preceding page. (See "Final LINE-COUNTER Setting Rule" 6a.) If Saved Next Group Integer was not so set, the body group shall be presented on the page to which the report is currently positioned. If Saved Next Group Integer was so set, the RWCS moves the Saved Next Group Integer into LINE-COUNTER and resets Saved Next Group Integer to zero. If then the value in LINE-COUNTER is less than the integer of the first absolute LINE NUMBER clause, the body group shall be presented on the page to which the report is currently positioned. Otherwise the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS presents the body group on that page.

4. First Print Line Position Rules

   a. The first print line of the body group is presented on the line number specified by the integer of its LINE NUMBER clause.

   b. If the value in LINE-COUNTER is equal to or greater than the line number specified by the FIRST DETAIL phrase of the PAGE clause, and if no body group has previously been presented on the page to which the report is currently positioned, then the first print line of the current body group is presented on the line immediately following the line indicated by the value contained in LINE-COUNTER.

      If the value in LINE-COUNTER is equal to or greater than the line number specified by the FIRST DETAIL phrase of the PAGE clause, and if a body group has previously been presented on the page to which the report is currently positioned, then the first print line of the current body group is presented on the line that is obtained by adding the contents of LINE-COUNTER and the integer of the first LINE NUMBER clause of the current body group.
      If the value in LINE-COUNTER is less than the line number specified by the FIRST DETAIL phrase of the PAGE clause, then the first print line of the body group is presented on the line specified by the FIRST DETAIL phrase.

   c. The body group is not presented.

   d. The sum of the contents of LINE-COUNTER and the integer of the first LINE NUMBER clause defines the line number on which the first print line is presented.

5. Next Group Rule

   The integer of the absolute NEXT GROUP clause must specify a line number that is not less than that specified in the FIRST DETAIL phrase of the PAGE clause, and that is not greater than that specified in the FOOTING phrase of the PAGE clause.

6. Final LINE-COUNTER Setting Rules

   a. If the body group that has just been presented is a CONTROL FOOTING report group and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

      For all other cases the RWCS makes a comparison of the line number on which the final print line of the body group was presented and the integer of the NEXT GROUP clause. If the former is less than the latter, then the RWCS places the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting. If the former is equal to or greater than the latter, then the RWCS places the line number specified by the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER setting; in addition the RWCS places the NEXT GROUP integer into the Saved Next Group Integer location.

b. If the body group that has just been presented is a CONTROL FOOTING report group, and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

   For all other cases the RWCS computes a trial sum in a work location. The trial sum is computed by adding the integer of the NEXT GROUP clause to the line number on which the final print line of body group was presented. If the sum is less than the line number specified by the FOOTING phrase of the PAGE clause, then the RWCS places that sum into LINE-COUNTER as the final LINE-COUNTER setting. If the sum is equal to or greater than the line number specified by the FOOTING phrase of the PAGE clause, then the RWCS places the line number specified by the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER setting.

c. If the body group that has just been presented is a CONTROL FOOTING report group, and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

   For all other cases the RWCS places the line number specified by the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER SETTING.

d. The final LINE-COUNTER setting is the number on which the final print line of the body group was presented.

e. LINE-COUNTER is unaffected by the processing of a non-printable body group.

f. If the body group that has just been presented is a CONTROL FOOTING report group, and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

   For all other cases the RWCS places the sum of the line number on which the final print line was presented and the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting.

8.13.5 PAGE FOOTING Presentation Rules

The table below points to the appropriate Presentation Rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a PAGE FOOTING report group.

Table 8-7. PAGE FOOTING Presentation Rules

		APPLICABLE RULES ***				
******		**If the PAGE clause is specified ******				
Sequence of LINE NUMBER clauses *	NEXT GROUP clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final Line-Counter Setting
A R	Absolute	1	2	3a	4a	5a
A R	Relative	1	2	3a	4b	5b
A R		1	2	3a		5c
				3b		5d

***** The meaning of the abbreviations used in column 1 has been previously stated. (See "LINE NUMBER Clause Notation", above.)

****** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the Report Group Description entry.

******* A blank entry in an Applicable Rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

******** If the PAGE clause is omitted from the Report Description entry, then a PAGE FOOTING report group may not be defined. (See the "TYPE Clause", Chapter 9).

Presentation Rules

1. Upper Limit Rule

   The first line number on which the PAGE FOOTING report group can be presented, is the line number obtained by adding one to the value of integer-5 of the FOOTING phrase of the PAGE clause.

2. Lower Limit Rule

   The last line number on which the PAGE FOOTING report group can be presented is the line number specified by integer-1 of the PAGE clause.

3. First Print Line Position Rules

   a. The First print line of the PAGE FOOTING report group is presented on the line specified by the integer of its LINE NUMBER clause.

   b. The PAGE FOOTING report group is not presented.

4. NEXT GROUP rules

   a. The NEXT GROUP integer must be greater than the line number on which the final print line of the PAGE FOOTING report group is presented. In addition, the NEXT GROUP integer must not be greater than the line number specified by integer-1 of the PAGE clause.

   b. The sum of the NEXT GROUP integer and the line number on which the final print line of the PAGE FOOTING report group is presented must not be greater than the line number specified by integer-1 of the PAGE clause.

5. Final LINE-COUNTER Setting Rules

   a. After the PAGE FOOTING report group is presented, the RWCS places the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting.

   b. After the PAGE FOOTING report group is presented, the RWCS places the sum of the NEXT GROUP integer and the line number on which the final print line of the PAGE FOOTING report group was presented into LINE-COUNTER as the final LINE-COUNTER setting.

   c. After the PAGE FOOTING report group is presented the final LINE-COUNTER setting is the line number on which the final print line of the PAGE FOOTING report group was presented.

   d. LINE-COUNTER is unaffected by the processing of a non-printable report group.

8.13.6 REPORT FOOTING Presentation Rules

The table below points to the appropriate Presentation Rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a REPORT FOOTING report group.

Table 8-8. REPORT FOOTING Presentation Rules

		APPLICABLE RULES ***						
******		**If the PAGE clause is specified**					**If the PAGE clause is omitted**	
Sequence of LINE NUMBER clauses *	NEXT GROUP clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final Line-Counter Setting	First Print Line Position	Final Line Counter Setting
A R		1a	2	3a		4a	Illegal Combination (see the LINE NUMBER clause)	
R		1a	2	3b		4a	3d	4a
NP R		1b	2	3c		4a	Illegal Combination (see the LINE NUMBER clause)	
				3e		4b	3e	4b

***** The meaning of the abbreviations used in column 1 has been previously stated. (See "LINE NUMBER Clause Notation", above.)

****** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the Report Group Description entry

******* A blank entry in an Applicable Rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

Presentation Rules

1. Upper Limit Rules

   a. If a PAGE FOOTING report group has been presented on the page to which the report is currently positioned, then the first line number on which the REPORT FOOTING report group can be presented is one greater than the final LINE-COUNTER setting established by the PAGE FOOTING report group.

      Otherwise the first line number on which the REPORT FOOTING report group can be presented is the line number obtained by adding one and the value of integer-5 of the PAGE clause.

   b. The first line number on which the REPORT FOOTING report group can be presented, is the line number specified by the HEADING phrase of the PAGE clause.

2. Lower Limit Rule

   The last line number on which the REPORT FOOTING report group can be presented is the line number specified by integer-1 of the PAGE clause.

3. First Print Line Position Rules

   a. The first print line of the REPORT FOOTING report group is presented on the line specified by the integer of its LINE NUMBER clause.

   b. If a PAGE FOOTING report group has been presented on the page to which the report is currently positioned, then the sum of the final LINE-COUNTER setting established by the PAGE FOOTING report group and the integer of the first LINE NUMBER clause of the REPORT FOOTING report group defines the line number on which the first print line of the REPORT FOOTING report group is presented. Otherwise the sum of the integer of the first LINE NUMBER clause of the REPORT FOOTING report group, and the line number specified by the value of integer-5 of the FO OTING phrase of the PAGE clause defines the line number on which the first line of the REPORT FOOTING report group is presented.

   c. The NEXT PAGE phrase in the first absolute LINE NUMBER clause directs that the REPORT FOOTING report group is presented on a page on which no other report group has been presented. The first print line of the REPORT FOOTING report group is presented on the line number specified by the integer of its LINE NUMBER clause.

   d. The sum of the contents of LINE-COUNTER and the integer of the first LINE NUMBER clause defines the line number on which the first print line is presented.

   e. The REPORT FOOTING report group is not presented.

4. Final LINE-COUNTER Setting Rules

   a. The final LINE-COUNTER setting is the line number on which the final print line of the REPORT FOOTING report group is presented.

   b. LINE-COUNTER is unaffected by the processing of a non-printable report group.

9. Data Division - Clauses

This chapter describes the clauses used in the Data Division. For ease of reference, they are presented in alphabetical order.

The following clauses are described in this chapter:

```
BLANK WHEN ZERO          NEXT GROUP
BLOCK CONTAINS           OCCURS
CODE                     PAGE
CODE-SET                 PICTURE
COLUMN NUMBER            RECORD
CONTROL                  REDEFINES
DATA-NAME/FILLER         RENAMES
DATA RECORDS             REPORT
EXTERNAL                 SIGN
GLOBAL                   SOURCE
GROUP INDICATE           SUM
JUSTIFIED                SYNCHRONIZED
LABEL RECORDS            TYPE
LEVEL-NUMBER             USAGE
LINAGE                   VALUE
LINE NUMBER              VALUE OF
```

9.1 BLANK WHEN ZERO

Description

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

Format

```
BLANK WHEN ZERO
```

Syntax

1. The BLANK WHEN ZERO clause can be specified only for an elementary item whose PICTURE is specified numeric or numeric edited. (See the "PICTURE Clause", this chapter.)

2. The numeric or numeric edited Data Description entry to which the BLANK WHEN ZERO clause applies, must be described either implicitly or explicitly as USAGE IS DISPLAY.

|3. This clause cannot be used for variable length items.|

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.

9.2 BLOCK CONTAINS

Description

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

Format

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

General Rules

1. When the CHARACTERS phrase is specified the physical record size is specified in terms of the number of character positions required to store the physical record regardless of the type of characters used to represent the items within the physical record. |If the file is implicitly or explicitly in SSF format, with records of fixed length, the user must take into account the 8 characters of the SSF header when calculating the size of the block.|

2. If integer-1 is not specified, integer-2 represents the exact size of the physical block. If integer-1 and integer-2 are both specified, they refer to the minimum and maximum sizes of the physical record respectively.

3. When the RECORDS option is selected, it specifies the size of a block in terms of logical records. The block size is integer-2 times the largest record defined within the FD, plus any block control characters that may be defined for that particular file organization.

4. The default for this clause is: BLOCK CONTAINS 1 RECORDS.

5. If the associated file connector is an external file connector, all BLOCK CONTAINS clauses in the run unit which are associated with that file connector must have the same value for integer-1 and integer-2.

9.3 CODE

Description

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

Format

```
CODE literal-1
```

Syntax Rules

1. Literal-1 must be a two-character non-numeric literal.

2. If the CODE clause is specified for any report in a file, then it must be specified for all reports in that file.

General Rules

1. When the CODE clause is specified, literal-1 is automatically placed in the first two character positions of each Report Writer logical record.

2. The positions occupied by literal-1 are not included in the description of the print line, but are included in the logical record size.

9.4 CODE-SET

Description

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

Format

```
                {   alphabet-name}
                { |------------| }
                { | NATIVE     | }
                { | STANDARD-1 | }
CODE-SET IS     { | STANDARD-2 | }
                { | ASCII      | }
                { | EBCDIC     | }
                { | GBCD       | }
                { | JIS        | }
                  |------------|
```

Syntax Rules

1. If 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 be specified for a file whose ORGANIZATION is not SEQUENTIAL.|

General Rules

1. If the CODE-SET clause is specified:

   a. Upon successful execution of an OPEN statement, the character set used to represent the data on the external media is the one referenced by alphabet-name, |NATIVE, STANDARD-1, STANDARD-2, ASCII, EBCDIC or JIS in the File Description entry associated with the file-name specified| in the OPEN statement (see the "SPECIAL NAMES Paragraph", Chapter 7).

   b. It specifies the algorithm for converting the character set on the external media from/to the native character set during the execution of an input or output operation.

|2. For an indexed file, the CODE-SET clause specifies the collating sequence. This is the sequence of values of a given key of reference used to process the file sequentially.|

3. If the CODE-SET clause is not specified, the native character code set is assumed for data on the external media, |except that if the ORGANIZATION is of ANSI type, CODE-SET IS ASCII is assumed.|

4. If the associated file connector is an external file connector, all CODE-SET clauses in the run unit which are associated with the same file connector must have the same character set.

|5. If the ORGANIZATION is of ANSI type, only CODE-SET IS STANDARD-1, CODE-SET IS ASCII, CODE-SET IS alphabet-name with the alphabet-name clause referencing STANDARD-1 or ASCII are permitted.|

6. CODE-SET IS |NATIVE or EBCDIC or| alphabet-name specified as NATIVE or EBCDIC means that the file code set is the EBCDIC character set.

7. CODE-SET IS |ASCII or STANDARD-1 or| alphabet-name specified as ASCII or STANDARD-1 means that the file code set is the ASCII character set.

   If the internal-file-name specified for the file in a SELECT clause of the Environment Division is suffixed by -TAPE, the file is expected to be an ANSI file unless an organization qualifier is explicitly specified.

8. CODE-SET IS |STANDARD-2 or| alphabet-name specified as STANDARD-2 means that the file code set is the International Reference Version of the ISO 7-bit code defined in International Standard 646, 7-bit Coded Character Set for Information Processing Interchange.

9. CODE-SET IS |GBCD or| alphabet-name specified as GBCD means that the file code set is the Honeywell Series 100/400/600 character set.

10. CODE-SET IS |JIS or| alphabet-name specified as JIS means that the file code set is the Japanese Industry Standard character set.

9.5 COLUMN NUMBER

Description

The COLUMN NUMBER clause identifies a printable item and specifies the position of the item on a print line.

Format

```
COLUMN NUMBER IS integer-1
```

Syntax Rules

1. The COLUMN NUMBER clause can be specified only at the elementary level within a report group. The COLUMN NUMBER clause, if present, must appear in or be subordinate to an entry that contains a LINE NUMBER clause.

2. Within a given print line, the printable items must be defined in ascending column number order such that each printable item defined occupies a unique sequence of contiguous character positions.

General Rules

1. The COLUMN NUMBER clause indicates that the object of a SOURCE clause or the object of a VALUE clause or the sum counter defined by a SUM clause is to be presented on the print line. The absence of a COLUMN NUMBER clause indicates that the entry is not to be presented on a print line.

2. Integer-1 specifies the column number of the leftmost character position of the printable item.

3. The Report Writer Control System supplies space characters for all positions of a print line that are not occupied by printable items.

4. The leftmost position of the print line is considered to be column number 1.

9.6 CONTROL

Description

The CONTROL clause establishes the levels of the control hierarchy for the report.

Format

```
{CONTROL IS  } {{data-name-1}...        }
{            } {                        }
{CONTROLS ARE} { FINAL [data-name-1]... }
```

Syntax Rules

1. Data-name-1 must not be defined in the Report Section. Data-name-1 may be qualified.

2. Each recurrence of data-name-1 must identify a different data item.

3. Data-name-1 must not have subordinate to it a variable-occurrence data item.

General Rules

1. Data-name-1 and the word FINAL specify the levels of the control hierarchy. FINAL, if specified, is the highest control, data-name-1 is the major control, the next recurrence of data-name-1 is an intermediate control, etc. The last recurrence of data-name-1 is the minor control.

2. The execution of the chronologically first GENERATE statement for a given report causes the RWCS to save the values of all control data items associated with that report. On subsequent executions of all GENERATE statements for that report, control data items are tested by the RWCS for a change of value. A change of value in any control data item causes a control break to occur. This control break is associated with the highest level for which a change of value is noted (see the "GENERATE Statement", Chapter 11).

3. The Report Writer Control system tests for a control break by comparing the contents of each control data item with the prior contents of each control data item that was saved when the previous GENERATE statement for the same report was executed. The RWCS applies the inequality relation test as follows:

   a. If the control data item is a numeric data item, the relation test is for the comparison of two numeric operands.

   b. If the control data item is an index data item, the relation test is for the comparison of two index data items.

   |c. If the control data item is a boolean data item, the relation test is for comparison of two boolean data items.

   d. If the control data item is a pointer data item, the relation test is for comparison of two pointer data items.|

   e. If the control data item is a data item other than as described in paragraph 3a, 3b, |3c and 3d| , the relation test is for the comparison of two non-numeric operands.

   The inequality relation test is further explained in the appropriate paragraph (see "Relation Condition", Chapter 10).

4. FINAL is used when the most inclusive control group in the report is not associated with a control data-name.

9.7 DATA-NAME/FILLER

Description

A data-name specifies the name of the data being described. The key word FILLER may be used to specify a data item which is not referenced explicitly.

Format

```
 {data-name}
[{         }]
 { FILLER  }
```

Syntax Rules

1. In the File, Working-Storage, |Constant,| Communication and Linkage Sections, a data-name or the key word FILLER, if either is specified, must be the first word following the level-number in each Data Description entry.

2. In the Report Section a data-name need not appear in a Data Description entry and the key word FILLER must not be used.

General Rules

1. If this clause is omitted, the data item being described is treated as though FILLER had been specified.

2. The key word FILLER may be used to name a data item. Under no circumstances can a FILLER item be referred to explicitly.

   However, the key word FILLER may be used to name a conditional variable because such use does not require explicit reference to the data item itself, but only to the value contained therein.

3. In the Report Section, data-name must be given in the following cases:

   a. When the data-name represents a report group to be referred to by a GENERATE or a USE statement in the Procedure Division.

   b. When reference is to be made to the sum counter in the Procedure Division or Report Section.

   c. When a DETAIL report group is referenced in the UPON phrase of the SUM clause.

   d. When the data-name is required to provide sum counter qualification.

9.8 DATA RECORDS

Description

The DATA RECORDS clause serves only as documentation for the names of data records within their associated file. The DATA RECORDS clause is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

Format

```
     {RECORD IS  }
DATA {           } {data-name-1}...
     {RECORDS ARE}
```

Syntax Rule

Data-name-1 is the name of a data record and must have a 01 level-number Record Description, with the same name, associated to it.

General Rule

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.

9.9 EXTERNAL

Description

The EXTERNAL clause specifies that a data record |or a non-contiguous data-item| or a file connector is external. The constituent data items and group data items of an external data record are available to every program in the run unit which describes that record.

Format

```
IS EXTERNAL
```

Syntax Rules

1. The EXTERNAL clause may be specified only in File Description entries |or in Record Description entries| or in level-77 Data Description entries, in the Working-Storage |or the Constant Section.|

2. In the same program, the data-name specified as the subject of the entry whose level-number is 01 |or 77| that includes the EXTERNAL clause must not be the same data-name specified for any other Data Description entry which includes the EXTERNAL clause.

3. |The VALUE clause may be used in any Data Description entry which includes, or is subordinate to an entry which includes the EXTERNAL clause.| The VALUE clause may be specified for condition-names entries associated with any Data Description entry which includes or is subordinate to an entry which includes the EXTERNAL clause.

General Rules

1. The data contained in the record named by the data-name clause is external and may be accessed and processed by any program in the run unit which describes and, optionally, redefines it subject to the following rules.

2. Within a run unit, if two or more programs describe the same external data record, each record-name of the associated Record Description entries must be the same, and the records must define the same number of standard data format characters. However, a program which describes an external record may contain a Data Description entry including the REDEFINES clause which redefines the complete external record, and this complete re-definition need not occur identically in other programs in the run unit. (See the "REDEFINES Clause", this chapter).

3. Use of the EXTERNAL clause does not imply that the associated file-name or data-name is a global name (see the "GLOBAL Clause", this chapter).

4. The file connector associated with this File Description entry is an external file connector.

9.10 GLOBAL

Description

The GLOBAL clause specifies that a data-name, a file-name |, a cd-name| or a report-name is a global name. A global name is available to every program contained within the program which declares it.

Format

```
IS GLOBAL
```

Syntax Rules

1. The GLOBAL clause may be specified only in Data Description entries whose level-number is 01 in the File Section or in the Working-Storage Section |or in the Communication Section or in the Linkage Section, File Description entries, Sort-Merge File Description entries, Communication Description entries| or Report Description entries.

2. In the same Data Division, the Data Description entries for any two data items for which the same data-name is specified must not include the GLOBAL clause.

3. If the SAME RECORD AREA clause is specified for several files, the Record Description entries or the File Description entries for these files must not include the GLOBAL clause.

General Rules

1. A data-name, file-name |, cd-name| or report-name described using a GLOBAL clause is a global name. All data-names subordinate to a global name are global names. All condition-names associated with a global name are global names.

2. A statement in a program contained directly or indirectly within a program which describes a global name may reference that name without describing it again.

3. If the GLOBAL clause is used in a Data Description entry which contains the REDEFINES clause, it is only the subject of that REDEFINES clause which possesses the global attribute.

9.11 GROUP INDICATE

Description

The GROUP INDICATE clause specifies that the associated printable item is presented only on the first occurrence of its report group after a control break or page advance.

Format

```
GROUP INDICATE
```

Syntax Rule

The GROUP INDICATE clause may only be specified in a DETAIL report group entry that defines a printable item.

General Rules

1. If a GROUP INDICATE clause is specified, it causes the SOURCE or VALUE clause to be ignored and spaces supplied, except:

   a. On the first presentation of the DETAIL report group in the report, or

   b. On the first presentation of the DETAIL report group after every page advance, or

   c. On the first presentation of the DETAIL report group after every control break.

2. If the Report Description entry specifies neither a PAGE clause nor a CONTROL clause, then a GROUP INDICATE printable item is presented the first time its DETAIL is presented after the INITIATE statement is executed. Thereafter, spaces are supplied for indicated items with SOURCE or VALUE clauses.

9.12 JUSTIFIED

Description

The JUSTIFIED clause permits alternate positioning of data within a receiving data item.

Format

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

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.

4. The JUSTIFIED clause may be specified only for a data item described, implicitly or explicitly, as USAGE IS DISPLAY.

|5. The JUSTIFIED clause must not be specified for a variable-length data item.|

General Rules

1. When the 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, and space fill |or boolean character zero fill in case of boolean data items,| is provided 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 Rules for Data Alignment", Chapter 3).

9.13 LABEL RECORDS

Description

The LABEL RECORDS clause specifies whether labels are present. The LABEL RECORDS clause is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

Format

```
       {RECORD IS  } {STANDARD}
[LABEL {           } {         }]
       {RECORDS ARE} {OMITTED  }
```

Syntax Rule

The LABEL RECORDS clauses specified in the File Description entries associated with file-names specified in a MULTIPLE FILE TAPE clause in the I-O-CONTROL paragraph must reflect a uniform labelling convention.

General Rules

1. OMITTED specifies that no explicit labels exist for the file or the device to which the file is assigned.

2. STANDARD specifies that labels exist for the file or the device to which the file is assigned, and the labels conform to DPS 7 label specifications.

3. If the file connector associated with this file is an external file connector (see the "EXTERNAL Clause", this chapter), all LABEL RECORDS clauses in the run unit which are associated with that file connector must have the same specification.

9.14 LEVEL-NUMBER

Description

The level-number indicates the position of a data item within the hierarchical structure of a logical record or report group. In addition, it is used to identify entries for Working-Storage items, Linkage items, |Constant items,| Condition-names and the RENAMES clause.

Format

```
level-number
```

Syntax Rules

1. A level-number is required as the first element in each Data Description entry (see "Record Description", Chapter 8).

2. Data Description entries subordinate to a CD, FD or SD entry must have level-numbers 01 through 49, 66 or 88.

3. Data Description entries subordinate to an RD entry must have level-numbers 01 through 49 only.

4. Data Description entries in the Working-Storage, |Constant| and Linkage Section must have level-numbers 01 through 49, 66, 77 or 88.

General Rules

1. The level-number 01 identifies the first entry in each Record Description or report group.

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

   a. Level-number 66 is assigned to identify RENAMES entries and can be used only as described by Format 2 of the Data Description skeleton.

   b. Level-number 77 is assigned to identify non-contiguous working storage data items, |non-contiguous constant data items| or non-contiguous linkage data items, and can be used only as described in Format 1 of the Data Description skeleton.

   c. Level-number 88 is assigned to entries that define condition-names associated with a conditional variable, and can be used only as described by Format 3 of the Data Description skeleton.

3. Multiple level 01 entries subordinate to any given level indicator, other than RD, represent implicit re-definitions of the same area.

9.15 LINAGE

Description

The LINAGE clause provides a means for specifying the depth of a logical page in terms of number of lines. It also provides for specifying the size of the top and bottom margins on the logical page, and the line number, within the page body, at which the footing area begins.

Format

```
          {data-name-1}
LINAGE IS {           } LINES
          {integer-1  }

                        {data-name-2}
       [WITH FOOTING AT {           }]
                        {integer-2  }

                     {data-name-3}
       [LINES AT TOP {           }]
                     {integer-3  }

                        {data-name-4}
       [LINES AT BOTTOM {           }]
                        {integer-4  }
```

Syntax Rules

1. Data-name-1, data-name-2, data-name-3, data-name-4 must reference elementary unsigned numeric integer data items.

2. Data-name-1, data-name-2, data-name-3, data-name-4 may be qualified.

3. Integer-2 must not be greater than integer-1.

4. Integer-3, integer-4 may be zero.

|5. If the LINAGE clause is applied to an external file, only integers are permitted in the clause, and must be the same in all programs of the run-unit where the file is declared.|

General Rules

1. The LINAGE clause provides a means for specifying the size of a logical page in terms of number of lines. The logical page size is the sum of the values referenced by each phrase except the FOOTING phrase. If the LINES AT TOP or LINES AT BOTTOM phrases are not specified, the values of these items are zero. If the FOOTING phrase is not specified, no end-of-page condition independent of the page overflow condition exists.

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

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

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

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

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

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

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

7. The values of the data items, referenced by data-name-1, data-name-3, and data-name-4, if specified, are used as follows:

   a. The values of the data items, at the time an OPEN statement with the OUTPUT phrase is executed for the file, are used to specify the number of lines that are to comprise each of the indicated sections for the first logical page.

   b. The values of the data items, at the time a WRITE statement with the ADVANCING PAGE phrase is executed or page overflow condition occurs (see the "WRITE Statement, Chapter 13), are used to specify the number of lines that are to comprise each of the indicated sections for the next logical page.

8. The value of the data item referenced by data-name-2, if specified, at the time an OPEN statement with the OUTPUT phrase is executed for the file, is used to define

the footing area for the first logical page. At the time a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs, it is used to define the footing area for the next logical page.

9. A LINAGE-COUNTER is generated by the presence of a LINAGE clause. The value in the LINAGE-COUNTER at any given time represents the line number at which the device is positioned within the current page body. The rules governing the LINAGE-COUNTER are as follows:

    a. A separate LINAGE-COUNTER is supplied for each file described in the File Section whose File Description entry contains a LINAGE clause.

    b. LINAGE-COUNTER may be referenced only in Procedure Division statements; however, only the Input-Output Control System may change the value of LINAGE-COUNTER. Since more than one LINAGE-COUNTER may exist in a program, the user must qualify LINAGE-COUNTER by file-name when necessary.

    c. LINAGE-COUNTER is automatically modified, according to the following rules, during the execution of a WRITE statement to an associated file:

        (i) When the ADVANCING PAGE phrase of the WRITE statement is specified, the LINAGE-COUNTER is automatically reset to one (1). During the resetting of LINAGE-COUNTER to the value one (1) the value of LINAGE-COUNTER is implicitly incremented to exceed the value specified by integer-1 or the data item referenced by data-name-1.
        (ii) When the ADVANCING identifier-2 or integer-1 phrase of the WRITE statement is specified, the LINAGE-COUNTER is incremented by integer-1 or the value of the data item referenced by identifier-2.
        (iii) When the ADVANCING phrase of the WRITE statement is not specified, the LINAGE-COUNTER is incremented by the value one (1). (see the "WRITE Statement", Chapter 13).
        (iv) The value of LINAGE-COUNTER is automatically reset to one (1) when the device is repositioned to the first line that can be written on for each of the succeeding logical pages (see the "WRITE Statement", Chapter 13).

    d. The value of LINAGE-COUNTER is automatically set to one (1) at the time an OPEN statement with the OUTPUT phrase is executed for the associated file.

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

11. If the file connector associated with this File Description entry is an external file connector, all File Description entries in the run unit which are associated with this file connector must have:

    a. A LINAGE clause if any File Description entry has a LINAGE clause.
    b. The same corresponding values for integer-1, integer-2, integer-3 and integer-4, if specified.
    c. The same corresponding external data items referenced by data-name-1, data-name-2, data-name-3 and data-name-4.

9.16 LINE NUMBER

Description

The LINE NUMBER clause specifies vertical positioning information for its report group.

Format

```
                {integer-1 [ON NEXT PAGE]}
LINE NUMBER IS  {                        }
                {PLUS integer-2          }
```

Syntax Rules

1. Integer-1 and integer-2 must not exceed three significant digits in length.

   Neither integer-1 nor integer-2 may be specified in such a way as to cause any line of a report group to be presented outside of the vertical subdivision of the page designated for that report group type, as defined by the PAGE clause (see the "PAGE Clause", this chapter).

   Integer-2 may be zero |except in the first LINE NUMBER clause that appears in a Report Group description.|

2. Within a given Report Group Description entry, an entry that contains a LINE NUMBER clause must not contain a subordinate entry that also contains a LINE NUMBER clause.

3. Within a given Report Group Description entry, all absolute LINE NUMBER clauses must precede all relative LINE NUMBER clauses.

4. Within a given Report Group Description entry, successive absolute LINE NUMBER clauses must specify integers that are in ascending order. The integers need not be consecutive.

5. If the PAGE clause is omitted from a given Report Description entry, only relative LINE NUMBER clauses can be specified in any Report Group Description entry within that report.

6. Within a given Report Group Description entry a NEXT PAGE phrase can appear only once and, if present, must be in the first LINE NUMBER clause in that Report Group Description entry.

7. A LINE NUMBER clause with the NEXT PAGE phrase can appear only in the description of body groups and in a REPORT FOOTING report group.

8. Every entry that defines a printable item (see the "COLUMN NUMBER Clause", this chapter) must either contain a LINE NUMBER clause, or be subordinate to an entry that contains a LINE NUMBER clause.

9. The first LINE NUMBER clause specified within a PAGE FOOTING report group must be an absolute LINE NUMBER clause.

General Rules

1. A LINE NUMBER clause must be specified to establish each print line of a report group.

2. The RWCS effects the vertical positioning specified by a LINE NUMBER clause, before presenting the print line established by that LINE NUMBER clause.

3. Integer-1 specifies an absolute line number. An absolute line number specifies the line number on which the print line is presented.

4. Integer-2 specifies a relative line number. If a relative LINE NUMBER clause is not the first LINE NUMBER clause in the Report Group Description entry, then the line number on which its print line is presented is determined by calculating the sum of the line number on which the previous print line of the report group was presented and integer-2 of the relative LINE NUMBER clause. If integer-2 is zero, the line will be printed on the same line as the previous print line.

   If a relative LINE NUMBER clause is the first LINE NUMBER clause in the Report Group Description entry, then the line number on which its print line is presented is determined by specified rules (see "Presentation Rules Tables", Chapter 8).

5. The NEXT PAGE phrase specifies that the report group is to be presented beginning on the indicated line number on a new page (see "Presentation Rules Tables", Chapter 8).

9.17 NEXT GROUP

Description

The NEXT GROUP clause specifies information for vertical positioning of a page following the presentation of the last line of a report group.

Format

```
                {integer-1     }
NEXT GROUP IS {PLUS integer-2}
                {NEXT PAGE     }
```

Syntax Rules

1. A report group entry must not contain a NEXT GROUP clause unless the description of that report group contains at least one LINE NUMBER clause.

2. Integer-1 and integer-2 must not exceed three significant digits in length.

3. If the PAGE clause is omitted from the Report Description entry only a relative NEXT GROUP clause may be specified in any Report Group Description entry within that report.

4. The NEXT PAGE phrase of the NEXT GROUP clause must not be specified in a PAGE FOOTING report group.

5. The NEXT GROUP clause must not be specified in a REPORT FOOTING report group or in a PAGE HEADING report group.

General Rules

1. Any positioning of the page specified by the NEXT GROUP clause takes place after the presentation of the report group in which the clause appears (see "Presentation Rules Tables", Chapter 8).

2. The Report Writer Control System uses the vertical positioning information supplied by the NEXT GROUP clause along with information from the TYPE and PAGE clauses, and the value in LINE-COUNTER, to determine a new value for LINE-COUNTER (see "Presentation Rules Tables", Chapter 12.

3. The NEXT GROUP clause is ignored by the RWCS when it is specified on a CONTROL FOOTING report group that is at a level other than the highest level at which a control break is detected.

4. The NEXT GROUP clause of a body group refers to the next body group to be presented, and therefore can affect the location at which the next body group is presented. The NEXT GROUP clause of a REPORT HEADING report group can affect the location at which the PAGE HEADING report group is presented. The NEXT GROUP clause of a PAGE FOOTING report group can affect the location at which the REPORT FOOTING report group is presented (see "Presentation Rules Tables", Chapter 8).

9.18 OCCURS

Description

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

Format 1

```
OCCURS integer-2 TIMES

    {ASCENDING }
   [{          } KEY IS {data-name-2}... ]...
    {DESCENDING}

    [INDEXED BY {index-name-1}... ]
```

Format 2

```
OCCURS integer-1 TO integer-2 TIMES DEPENDING ON data-name-1

    {ASCENDING }
   [{          } KEY IS {data-name-2}... ]...
    {DESCENDING}

    [INDEXED BY {index-name-1}... ]
```

Syntax Rules

1. The OCCURS clause must not be specified in a data description entry that:

   a. Has a 01, 66, 77 or an 88 level-number, or

   b. Has a variable occurrence data-item subordinate to it.

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

3. The first specification of data-name-2 must be the name of either the entry containing the OCCURS clause or an entry subordinate to the entry containing the OCCURS clause. Subsequent specification of data-name-2 must be subordinate to the entry containing the OCCURS clause.

4. Data-name-2 must be specified without the subscripting normally required.

5. Where both integer-1 and integer-2 are used, integer-1 must be greater than or equal to zero and integer-2 must be greater than integer-1.

6. Data-name-1 must describe an integer.

|7. Data-name-1 and data-name-2 must not reference a boolean or a pointer data item.|

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

9. If the OCCURS clause is specified in a Data Description entry included in a Record Description entry containing the EXTERNAL clause, data-name-1, if specified, must reference a data item possessing the external attribute which is described in the same Data Division.

10. If the OCCURS clause is specified in a Data Description entry subordinate to one containing the GLOBAL clause, data-name-1, if specified, must be a global name and must reference a data item which is described in the same Data Division.

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

12. The data item identified by data-name-2 must not contain an OCCURS clause except when data-name-2 is the subject of the entry.

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

14. An INDEXED BY phrase is required if the subject of this entry, or an entry subordinate to this entry, is to be referenced 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.

15. Index-name-1 must be a unique word within the program.

16. The INDEXED BY clause must not be specified if integer-2 is greater than 65500 nor must it be specified if one occurrence of the data item described in the entry containing the OCCURS clause is larger than 65500 bytes.

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 SEARCH or 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 "Subscripting Indexing Identifier", Chapter 3).

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.

3. The number of occurrences of the subject entry is defined as follows:

   a. In Format 1, the value of integer-2 represents the exact number of occurrences.

   b. In Format 2, the current value of the data item referenced by data-name-1 represents the number of occurrences.

      This format specifies that the subject of this entry has a variable number of occurrences. The value of integer-2 represents the maximum number of occurrences and the value of integer-1 represents the minimum number of occurrences. This does not imply that the length of the subject of the entry is variable, but that the number of occurrences is variable.

      At the time the subject of the entry is referenced or any data item subordinate or superordinate to the subject of the entry is referenced, the value of the data item referenced by data-name-1 must fall within the range integer-1 through integer-2. The contents of the data item whose occurrence numbers exceed the value of the data item referenced by data-name-1 are undefined.

4. When a group item, having subordinate to it an entry that specifies Format 2 of the OCCURS clause, is referenced, the part of the table area used in the operation is determined as follows:

   a. If the data item referenced by data-name-1 is outside the group, only that part of the table area that is specified by the value of the data item referenced by data-name-1 at the start of the operation will be used.

   b. If the data item referenced by data-name-1 is included in the same group and the group data item is referenced as a sending item, only that part of the sending area that is specified by the value of the data item referenced by data-name-1 at the start of the operation will be used. If the group is a receiving item, the maximum length of the group will be used.

5. When the KEY IS phrase is specified, the repeated data must be arranged in ascending or descending order according to the values contained in each data-name-2. The ascending or descending order is determined according to the rules for comparison of operands (see "Comparison of Numeric Operands" and "Comparison of Non-numeric Operands", Chapter 10). The data-names are listed in their descending order of significance.

6. If Format 2 is specified in a Record Description entry and if the associated File Description or Sort-Merge Description implies that records are variable length, then if the DEPENDING ON phrase of the RECORD clause is not specified, the contents of the data item referenced by data-name-1 of the OCCURS clause must be set to the number of occurrences to be written before the execution of any RELEASE, REWRITE, or WRITE statement.

9.19 PAGE

Description

The PAGE clause defines the length of a page and the vertical subdivisions within which report groups are presented.

Format

```
     [LIMIT IS  ]           [LINE ]
PAGE [          ] integer-1 [     ]
     [LIMITS ARE]           [LINES]

    [HEADING integer-2]

    [FIRST DETAIL integer-3]

    [LAST DETAIL integer-4]

    [FOOTING integer-5]
```

Syntax Rules

1. The HEADING, FIRST DETAIL, LAST DETAIL, and FOOTING phrases may be written in any order.

2. Integer-1 must not exceed three (3) significant digits in length.

3. Integer-2 must be greater than or equal to one (1).

4. Integer-3 must be greater than or equal to integer-2.

5. Integer-4 must be greater than or equal to integer-3.

6. integer-5 must be greater than or equal to integer-4.

7. Integer-1 must be greater than or equal to integer-5.

8. The following rules indicate the vertical subdivision of the page in which each TYPE of report group may appear when the PAGE clause is specified (see "Page Regions", below).

   a. A REPORT HEADING report group that is to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-1, inclusive.

      A REPORT HEADING report group that is not to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-3 minus 1, inclusive.

   b. A PAGE HEADING report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line

number specified by integer-2 to the line number specified by integer-3 minus 1, inclusive.

c. A CONTROL HEADING or DETAIL report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-3 to the line number specified by integer-4, inclusive.

d. A CONTROL FOOTING report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-3 to the line number specified by integer-5, inclusive.

e. A PAGE FOOTING report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-5 plus 1 to the line number specified by integer-1, inclusive.

f. A REPORT FOOTING report group that is to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-1, inclusive.

A REPORT FOOTING report group that is not to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-5 plus 1 to the line number specified by integer-1 inclusive.

9. All report groups must be described such that they can be presented on one page. The RWCS never splits a multi-line report group across page boundaries.

General Rules

1. The vertical format of a report page is established using the integer values specified in the PAGE clause.

   a. Integer-1 defines the size of a report page by specifying the number of lines available on each page.

   b. HEADING integer-2 defines the first line number on which a REPORT HEADING or PAGE HEADING report group may be presented.

   c. FIRST DETAIL integer-3 defines the first line number on which a body group may be presented. REPORT HEADING (without NEXT GROUP NEXT PAGE) and PAGE HEADING report groups may not be presented on or beyond the line number specified by integer-3.

   d. LAST DETAIL integer-4 defines the last line number on which a CONTROL HEADING or DETAIL report group may be presented.

   e. FOOTING integer-5 defines the last line number on which a CONTROL FOOTING report group may be presented. REPORT FOOTING (without LINE integer-1 NEXT PAGE) and PAGE FOOTING report groups must follow the line number specified by integer-5.

2. If the PAGE clause is specified the following implicit values are assumed for any omitted phrases:

a. If the HEADING phrase is omitted, a value of one (1) is assumed for integer-2.

b. If the FIRST DETAIL phrase is omitted, a value equal to integer-2 is given to integer-3.

c. If the LAST DETAIL and the FOOTING phrases are both omitted, the value of integer-1 is given to both integer-4 and integer-5.

d. If the FOOTING phrase is specified and the LAST DETAIL phrase is omitted, the value of integer-5 is given to integer-4.

e. If the LAST DETAIL phrase is specified and the FOOTING phrase is omitted, the value of integer-4 is given to integer-5.

3. If the PAGE clause is omitted, the report consists of a single page of indefinite length.

4. The presentation rules for each TYPE of report group are specified in the appropriate paragraph (see "Presentation Rules Tables", Chapter 8).

Page Regions

Page regions that are established by the PAGE clause are described below:

Table 9-1. Page Regions

Report Groups that may be presented in the region	First Line Number of the region	Last Line Number of the region
REPORT HEADING described with NEXT GROUP NEXT PAGE REPORT FOOTING described with LINE integer-1 NEXT PAGE	integer-2	integer-1
REPORT HEADING not described with NEXT GROUP NEXT PAGE PAGE HEADING	integer-2	integer-3 minus 1
CONTROL HEADING DETAIL	integer-3	integer-4
CONTROL FOOTING	integer-3	integer-5
PAGE FOOTING REPORT FOOTING not described with LINE integer-1 NEXT PAGE	integer-5 plus 1	integer-1

9.20 PICTURE

Description

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

Format

```
{PICTURE }                      |--------------------------|
{        } IS character-string | [DEPENDING ON data-name] |
{PIC     }                      |--------------------------|
```

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 lower-case letters corresponding to the uppercase letters representing the PICTURE symbols A, B, |E, L,| P, S, V, X, Z, CR and DB are equivalent to their uppercase representations in a PICTURE character string. All other lower-case letters are not equivalent to their corresponding uppercase representation.

4. The maximum number of characters allowed in the character-string is 30.

|5. The DEPENDING ON clause must not be specified in the Report Section.

6. Data-name must describe an elementary integer. It must not be defined in the Report Section. Data-name may be qualified.|

7. The PICTURE clause must be specified for every elementary item except an index data item, |a data item whose USAGE is COMPUTATIONAL-9, COMPUTATIONAL-10, COMPUTATIONAL-15 or POINTER| or the subject of the RENAMES clause. In these cases the use of this clause is prohibited. |It may be omitted for a data-item whose USAGE is COMPUTATIONAL-1 or COMPUTATIONAL-2.|

8. PIC is an abbreviation for PICTURE.

9. The asterisk when used as the Zero Suppression symbol and the clause BLANK WHEN ZERO may not appear in the same entry.

|10. The DEPENDING ON clause must be specified if, and only if, the PICTURE character-string contains the character 'L'.

11. The DEPENDING ON clause must not appear in a Data Description entry which contains a JUSTIFIED clause.|

General Rules

1. There are six categories of data that can be described with a PICTURE clause: |boolean,| alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited.

|2. To define an item as boolean:

   a. The PICTURE character string can only contain the symbol '1' and

   b. Its contents, when represented in Standard Data Format, must be a combination of the numeric characters '0' and '1'.|

3. To define an item as alphabetic:

   a. The PICTURE character-string can contain only the symbols 'A' |, 'L';| and

   b. Its contents when represented in Standard Data Format must be one or more alphabetic characters.

4. To define an item as numeric:

   a. When it is fixed point:

   (1) The PICTURE character-string can contain only 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 |(or 1 to 30 if the compilation is run with LEVEL=NSTD)|; and

   (2) If unsigned, its contents when represented in Standard Data Format must be one or more numeric characters; if signed, the item may also contain a '+', '-', or other representation of an operational sign (see the "SIGN Clause", this chapter).

|b. When it is floating-point:

(1) Its PICTURE character-string must be of the form:

```
|-------------|
| [S] kVm ESn |
|-------------|
```

where:

(a) The brackets indicate that the presence of the symbol 'S' is optional.

(b) The symbols 'k' and 'm' represent the significand. Each represents zero, one or more occurrences of the symbol '9'. The significand must contain at least one '9'. The number of digit positions that can be described by the PICTURE character-string of the significand must range from 1 to 30 inclusive.

(c) The symbol 'E' indicates the use of floating-point representation and is not counted in the size of the item and does not appear internally.

(d) The symbol 'n' represents the digits of the exrad. It consists of one or more occurrences of the symbol '9'. The number of digit positions that can be described by the PICTURE character-string of the exrad must at least be one but not more than two.

(2) If unsigned, the significand, when represented in Standard Data Format, must contain one or more numeric characters. If signed, the significand must contain a leading separate sign character followed by one or more numeric characters.

(3) The exrad must be signed and, when represented in Standard Data Format, must contain a leading separate sign character followed by one or more numeric characters.|

5. To define an item as alphanumeric:

a. Its PICTURE character-string is restricted to certain combinations of the symbols |'A', 'L', '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 only 'A's and the symbol 'L')| or all '9's does not define an Alphanumeric item, and,

b. Its contents when represented in Standard Data Format must be one or more characters in the computer's character set.

6. To define an item as alphanumeric edited:

a. Its PICTURE character-string is restricted to certain combinations of the following symbols: 'A', 'X', '9', 'B', '0'(zero), and '/' (slant); and must contain at least one 'A' or 'X' and must contain at least one 'B' or '0' (zero) or '/' (slant).

b. Its contents when represented in Standard Data Format must be one or more characters in the computer's character set.

7. To define an item as numeric edited:

   a. When it is fixed-point:

   (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; and

   (a) The number of digit positions that can be represented in the PICTURE character-string must range from 1 to 18 inclusive |(or 1 to 30 if the compilation is run with LEVEL=NSTD)| ; and

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

   (2) The content of each of the character position must be consistent with the corresponding PICTURE symbol.

   |b. When it is floating-point:

   (1) Its PICTURE character-string must be in the form:|

```
|-------------|
| {+}         |
| { }K.mE + n |
| {-}         |
|-------------|
```

   |Where:

   (a) The braces indicate that a positive or negative sign must be specified.

   (b) The symbols 'k' and 'm' represent the significand. Each represents zero, one or more occurrences of the symbol '9'. The significand must contain at least one '9'. The number of digit positions that can be described by the PICTURE character-string must range from 1 to 30 inclusive.

   (c) The symbol 'E' is an insertion character delimiting the exrad and is included in the size of the data item.

   (d) The symbol 'n' represents the exrad. It consists of one or two occurrences of the symbol '9'.

   (2) The size of a floating-point numeric edited data item is: 4 + k + m + n. The minimum size is six characters.|

8. 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 unsigned non-zero integer which is enclosed in parentheses following the symbols 'A', ',', 'X', '9',|'1',| '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: |'E', 'L',| 'V', '.', 'CR', and 'DB'. The symbol 'S' may appear |twice in a floating-point numeric PICTURE character-string but| only once in any fixed-point numeric PICTURE character-string.

   |The number of valid character positions contained within a variable-length data item may vary. (See "General Rule" 9, the PICTURE symbol 'L'.)|

9. The function 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 an alphabetic character and is counted in the size of the item.

   B Each 'B' in the character-string represents a character position into which the space character will be inserted and is counted in the size of the item.

   |E The 'E' indicates that the symbols that follow to the right in the character-string represent the exrad of a floating-point numeric or numeric-edited data item. The 'E' may appear only once in a PICTURE character-string. When used in the character-string of a floating-point numeric-edited data item, the 'E' represents the character-position into which the character 'E' is inserted. Each 'E' is counted in the size of the data item being described. When used in the character-string of a floating-point numeric data item, the 'E' is not counted in the size of the item.

   L The 'L' is used to define a variable-length data item; and, when present, must appear as the first symbol in the PICTURE character-string.

      The 'L' can be used to describe a data item of the category alphanumeric or alphabetic. While the size of the data item is fixed, the number of valid character positions contained in the data item varies. The PICTURE character-string describes the fixed size of the data item; the 'L' is not counted in determining the size of the data item.

      When referenced in the Procedure Division, the data item is considered to contain a number of valid character positions equal to the value of the data item referenced by data-name. The content of any character positions in excess of the number specified by data-name is undefined. The valid character positions are contiguous and begin at the leftmost character position in the data item.

      When such a data item is explicitly referenced, or implicitly referenced by an INITIALIZE statement or a receiving field by a statement with the CORRESPONDING phrase, any character positions participate in the operation.

During the execution of any statement which explicitly or implicitly references a variable-length data item, the value of the data item referenced by data-name must be within the range one (1) through the size of the data item as defined by the PICTURE character-string.

If a group item which contains a variable-length data item is referenced, all character positions in the variable-length data item participate in the operation; i.e., the variability of the valid contents of the data item(s) is ignored, as is its USAGE, category, synchronization, etc.|

P Each 'P' in the character-string 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, or 30 if the compilation is run with LEVEL=NSTD)| in numeric edited items or numeric items. The scaling position character 'P' can appear only as a continuous string of 'P's in the leftmost or rightmost digit positions within a PICTURE character-string; since the scaling position character 'P' implies an assumed decimal point (to the left of 'P's if 'P's are leftmost PICTURE symbols and to the right if 'P's are rightmost PICTURE symbols), 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.

In certain operations that reference a data item whose PICTURE character-string contains the symbol 'P', the algebraic value of the data item is used rather than the actual representation of the data item. This algebraic value assumes the decimal point in the prescribed location and zero in place of the digit position specified by the symbol 'P'. The size of the value is the number of digit positions represented by the PICTURE character-string. These operations are any of the following:

a. Any operation requiring a numeric sending operand.

b. A MOVE statement where the sending operand is numeric and its PICTURE character-string contains the symbol 'P'.

c. A MOVE statement where the sending operand is numeric-edited and its PICTURE character-string contains the symbol 'P' and the receiving operand is numeric or numeric-edited.

d. A comparison where both operands are numeric.

In all other operations the digit positions specified with the symbol 'P' are ignored and are not counted in the size of the operand.

S For a fixed-point data item, the '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 character-string. the 'S' is not counted in determining the size (in terms of Standard Data Format characters) of the elementary data item unless the entry is subject to a SIGN clause which specifies the optional SEPARATE CHARACTER phrase (see the "SIGN Clause", this chapter).

|For a floating-point numeric item, the 'S' in a PICTURE character-string indicates the presence of a leading separate sign character on the significand and exrad. It must be written as the leftmost character in the PICTURE character-string of the significand and immediately following the 'E' which precedes the PICTURE character-string of the exrad. Each occurrence of the 'S' is counted as one character in determining the size (in terms of Standard Data Format characters) of the elementary data item.|

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 representing a digit position or scaling position, 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 and is counted in the size of the item.

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

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

|1 Each '1' in the character-string represents a boolean position which contains a boolean character. Each '1' is counted in the size of the data item being described.|

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

/ Each '/' (slant) in the character-string represents a character position into which the slant 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.

. When the character '.' (period) appears in the character-string it is an editing symbol which represents the decimal point for alignment purposes and in addition, represents a character position into which the character '.' will be inserted. The character '.' is counted in the size of the item. For a given program the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange the rules for the period apply to the comma and the rules for the comma apply to the period wherever they appear in a PICTURE clause.

+ - 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 is placed. |Except for a FLOATING- POINT PICTURE character-string, within which both '+' and '-' may appear,| these symbols are mutually exclusive in any given character-string, and each character used in the symbol is counted in determining the size of the data item. Both CR and DB will be uppercase in the receiving data item.

***** Each '*' (asterisk) in the character-string represents a leading numeric character position into which an asterisk is placed when the content of that position is a leading 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 either by the currency sign (see "COBOL Character Set", Chapter 3) or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph (Chapter 7). The currency symbol is counted in the size of the item.

9.20.1 Editing Rules

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

    a. Simple insertion

    b. Special insertion

    c. Fixed insertion

    d. Floating insertion.

    There are two types of suppression and replacement editing:

    a. Zero suppression and replacement with spaces

    b. Zero suppression and replacement with asterisks.

11. The type of editing which may be performed upon an item is dependent upon the category to which the item belongs. The table below specifies which type of editing may be performed upon a given category.

Table 9-2. Categories of Data and Editing

CATEGORY	Type of Editing
Alphabetic	None
Boolean	None
Numeric	None
Alphanumeric	None
Alphanumeric edited	Simple insertion '0', 'B' and '/'
Numeric edited	All, subject to General Rule 12

12. 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. |Neither floating insertion editing nor editing by zero suppression and replacement may be applied to a floating-point numeric edited item.|

13. Simple Insertion Editing: the ',' (comma) 'B' (space) ['E'] '0' (zero) and '/' (slant) 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. If the insertion character ',' (comma) is the last symbol in the PICTURE character-string, the PICTURE clause must be the last clause of the Data Description entry and must be immediately followed by the separator period (.). This results in the combination of ',.' appearing in the Data Description entry or, if the DECIMAL POINT IS COMMA clause is used, in two consecutive periods.

14. 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. If the insertion character is the last symbol in the PICTURE character-string, the PICTURE clause must be the last clause of that Data Description entry and must be immediately followed by the separator period (.). This results in two consecutive periods appearing in the Data Description entry, or the combination of ',.' if the DECIMAL POINT IS COMMA clause is used. 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.

15. 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' and '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. If these character positions contain the symbols 'CR' or 'DB', the uppercase letters are the insertion characters. 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 position to be counted in the size of the item except that it can be preceded by either a '+' or a '-' symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the PICTURE character-string.

    Editing sign control symbols produce the results shown in the table below depending upon the value of the data item.

Table 9-3. Results of Sign Control Symbols in Editing

Editing Symbol in Picture Character String	Result	
	Date Item positive or zero	Data Item Negative
+	+	-
-	space	-
CR	two spaces	CR
DB	two spaces	DB

16. 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 the character-string by a string of at least two of the floating insertion characters. This string of floating insertion characters may contain any of the simple insertion symbols or have simple insertion characters immediately to the right of this string. These simple insertion characters are part of the floating string. When the floating insertion character is the currency symbol, this string of floating insertion characters may have the fixed insertion characters 'CR' and 'DB' immediately to the right of this string.

    The leftmost character of the floating insertion string represents the leftmost limit of the floating symbols 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 character positions 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 the numeric character positions in the PICTURE character string are represented by the insertion characters, at least one of the insertion characters must be to the left of the decimal point.

    When the floating insertion character is the editing control symbol '+' or '-', the character inserted depends on the value of the data item.

Editing Symbol in Picture Character String	Result: Date Item positive or zero	Result: Data Item Negative
+	+	-
-	space	-

    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 character position for the floating insertion character. If the truncation does occur, the value of the data that is used for editing is the value after truncation (see the "Standard Rules for Data Alignment", Chapter 3).

17. Zero Suppression Editing: the suppression of leading zeros in numeric character positions is indicated by 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; 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 leading 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 either 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, including any editing characters, is spaces. If the value is zero and the suppression symbol is '*', the entire data item, including any insertion editing symbol except the actual decimal point will be '*'. In this case, the actual decimal point will appear in the data item.

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

9.20.2 Precedence Rules

19. The table "Picture Character Precedence Chart" below 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 (but not necessarily immediately), 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'.

20. At least one of the symbols 'A', 'X', 'Z',|'1',| '9' or '*', or at least two of the symbols '+', '-' or 'cs', must be present in a PICTURE character-string.

21. Non-floating insertion symbols '+' and '-', floating insertion symbols 'Z', '*', '+', '-' and 'CS', and the symbol 'P' appear twice in the table "Picture Character Precedence Chart" below. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position |or immediately following the symbol 'E'.| The second appearance of the symbol in the chart represents its use to the right of the decimal point position.

Table 9-4. Picture Character Precedence Chart

Second Symbol	First Symbol	Non-Floating Insertion Symbols										Floating Insertion Symbols							Other Symbols						
		B	0	/	,	.	+ -	+ -	CR DB	cs	E	Z	Z	+ -	+ -	cs	cs	9	A X	L	S	V	P	P	1
Non Floating Insertion Symbols	B	X	X	X	X	X	X			X		X	X	X	X	X	X	X	X			X		X	
	0	X	X	X	X	X	X			X		X	X	X	X	X	X	X	X			X		X	
	/	X	X	X	X	X	X			X		X	X	X	X	X	X	X	X			X		X	
	,	X	X	X	X	X	X			X		X	X	X	X	X	X	X				X		X	
	.	X	X	X	X		X			X		X		X		X		X							
	+										X														
	{ + - }	X	X	X	X	X				X		X	X			X	X	X				X	X	X	
	{CR DB}	X	X	X	X	X				X		X	X			X	X	X				X	X	X	
	cs						X																		
	E					X	X											X			X	X			
Floating Insertion Symbols	{ Z * }	X	X	X	X		X			X		X													
	{ Z * }	X	X	X	X	X	X			X		X	X									X		X	
	{ + - }	X	X	X	X					X				X											
	{ + - }	X	X	X	X	X				X				X	X							X		X	
	cs	X	X	X	X		X									X									
	cs	X	X	X	X	X	X									X	X					X		X	
Others Symbols	9	X	X	X	X	X	X			X	X	X		X		X		X	X	X	X	X		X	
	{ A X }	X	X	X														X	X	X					
	L																								
	S										X														
	V	X	X	X	X		X			X		X		X		X		X			X		X		
	P	X	X	X	X		X			X		X		X		X		X			X		X		
	P						X			X											X	X		X	
	1																								X

9.21 RECORD

Description

The RECORD clause specifies the size of data records.

Format 1

```
RECORD CONTAINS integer-1 CHARACTERS
```

Format 2

```
RECORD IS VARYING IN SIZE [[FROM integer-2]

         [TO INTEGER-3][DEPENDING ON data-name-1]
```

Format 3:

```
                |---|               |---|
RECORD CONTAINS | [ | integer-4 TO  | ] | integer-5 CHARACTERS
                |---|               |---|
           |-------------------------------|
           | [DEPENDING ON data-name-2]    |
           |-------------------------------|
```

Syntax Rules

|1. If the RECORD clause is not specified and the REPORT clause is specified in the File Description entry for the file, the following implicit RECORD clause is provided:|

RECORD CONTAINS 1 TO 132 CHARACTERS.

Format 1

2. No Record Description entry for the file may specify a number of character positions greater than integer-1.

3. If the REPORT clause is specified in the File Description entry for the file, integer-1 must be greater than or equal to the length of the longest print line augmented by two (2) if the CODE clause is specified for the reports.

Format 2

4. Record Descriptions for the file must not describe records which contain a lesser number of character positions than that specified by integer-2 nor records which contain a greater number of character positions than that specified by integer-3.

5. If the REPORT clause is specified in the File Description entry for the file, the format 2 of the RECORD clause is not allowed.

6. Integer-3 must be greater than integer-2.

7. In a Sort-Merge Description entry, all the sort or merge keys specified in any SORT or MERGE statement referencing a sort or merge file must be contained within the first integer-2 number of character positions.

8. For an indexed file, all the record keys for the file must be contained within the first integer-2 number of character positions.

9. Data-name-1 must describe an elementary unsigned integer in the Working-Storage or Linkage section.

Format 3:

|10. Data-name-2 must describe an elementary unsigned integer in the File, Working-Storage or Linkage Section.

11. The DEPENDING ON clause in format 3 is kept for compatibility with previous releases.|

12. If the REPORT clause is specified in the File Description entry for the file, |the DEPENDING ON phrase must not be used,| integer-5 specifies the maximum record size for the file; integer-5 must be greater than or equal to the length of the longest print line, augmented by two (2) if the CODE clause is specified for the reports.

General Rules

All Formats

1. If the RECORD clause is not specified, the size of each data record is completely defined in the Record Description entry.

2. If the associated file connector is an external file connector, all File Description entries in the run unit which are associated with that file connector must specify the same values for integer-1 or integer-2, and integer-3. If the RECORD clause is not specified, all Record Description entries associated with this file connector must be the same length.

Format 1

3. Format 1 is used to specify fixed-length records. Integer-1 specifies the number of character positions contained in each record of the file. |However, the use of the VLR phrase of the SELECT clause overrides the use of format 1.|

Format 2

4. Format 2 is used to specify variable length records. Integer-2 specifies the minimum number of character positions to be contained in any record of the file. Integer-3 specifies the maximum number of character positions to be contained in any record of the file. |However, the use of the FLR phrase of the SELECT clause overrides the use of format 2.|

5. The number of character positions associated with a Record Description is determined by the sum of the number of character positions in all elementary data items excluding re-definitions and renamings, plus any implicit FILLER due to synchronization. If a table is specified:

    a. The minimum number of table elements described in the record is used in the summation above to determine the minimum number of character positions associated with the Record Description.

    b. The maximum number of table elements described in the record is used in the summation above to determine the maximum number of character positions associated with the Record Description.

6. If integer-2 is not specified, the minimum number of character positions to be contained in any record of the file is equal to the least number of character positions described for a record in that file.

7. If integer-3 is not specified, the maximum number of character positions to be contained in any record of the file is equal to the greatest number of character positions described for a record in that file.

8. If data-name-1 is specified, the number of character positions in the record must be placed into the data item referenced by data-name-1 before any RELEASE, REWRITE or WRITE statement is executed for that file.

9. If data-name-1 is specified, the execution of a DELETE, RELEASE, REWRITE, START or WRITE statement or the unsuccessful execution of a READ or RETURN statement does not alter the contents of the data item referenced by data-name-1.

10. During the execution of a RELEASE, REWRITE or WRITE statement, the number of character positions in the record is determined by the following conditions:

    a. If data-name-1 is specified, by the contents of the data item referenced by data-name-1.

    b. If data-name-1 is not specified and the record does not contain a variable occurrence data item, by the number of character positions in the record.

    c. If data-name-1 is not specified and the record does contain variable occurrence data items, by the sum of the fixed portion and that portion of the table described by the number of occurrences at the time of execution of the output statement.

11. If data-name-1 is specified, after the successful execution of a READ or RETURN statement for the file, the contents of the data item referenced by data-name-1 will indicate the number of character positions in the record just read.

12. If the INTO phrase is specified in the READ or RETURN statement, the number of character positions in the current record that participates as the sending data items in the implicit MOVE statement, is determined by the following conditions:

    a. If data-name-1 is specified, by the contents of the data item referenced by data-name-1.

    b. If data-name-1 is not specified, by the value that would have been moved into the data item referenced by data-name-1 had data-name-1 been specified.

Format 3

13. Format 3 of the RECORD clause specifies variable length records. Integer-4 and integer-5 refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively. |However, the use of the FLR phrase of the SELECT clause overrides the use of format 2.|

14. The size of each data record is specified in terms of the number of character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of characters in all fixed-length elementary items plus the sum of the maximum number of characters in any variable-length item subordinate to the record. This sum may be different from the actual size of the record (see "Data Types", Chapter 3, and the "SYNCHRONIZED Clause" and the "USAGE Clause", this chapter).

15. |If data-name-2 is specified, the number of character positions in the record must be placed into the data item referenced by data-name-2 before any RELEASE, REWRITE or WRITE statement is executed for the file. After the successful execution of a READ or RETURN statement for the file, the contents of the data item referenced by data-name-2 will indicate the number of character positions in the record just read. The execution of a DELETE or START statement, or the unsuccessful execution of a READ or RETURN statement, does not alter the contents of the data item referenced by data-name-2.|

16. During the execution of a RELEASE, REWRITE or WRITE statement, the number of character positions in the record is determined by the following conditions:

    |a. If data-name-2 is specified, by the contents of the data item referenced by data-name-2.|

    b. If |data-name-2 is not specified and| the record does not contain a variable occurrence data item, by the number of character positions in the record.

    c. If |data-name-2 is not specified and| the record does contain a variable occurrence data item, by the sum of that fixed portion and that portion of the table described by the number of occurrences at the time of the execution of the output.

       If the number of character positions in the logical record to be written is less than integer-2 or greater than integer-3, the output statement is unsuccessful and, except during execution of a RELEASE statement, the associated I-O Status is set to a value indicating the cause of the condition. (See "I-O Status".)

9.22 REDEFINES

Description

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

Format

```
             [data-name-1]
level-number [           ] REDEFINES data-name-2
             [FILLER     ]
```

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

Syntax Rules

1. The REDEFINES clause, when specified, must immediately follow the subject of the entry.

2. The level-number of data-name-2 and the subject of the entry must be identical, but must not be 66 or 88.

3. This clause must not be used in level 01 entries in the File Section.

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. However, data-name-2 may be subordinate to an item whose Data Description entry contains an OCCURS clause. In this case, the reference to data-name-2 in the REDEFINES clause may not be subscripted. Neither the original definition, nor the re-definition can include a variable occurrence data item.

6. If the data item referenced by data-name-2 is either declared to be an external data record or is specified with a level-number other than 01, the number of character positions it contains must be greater than or equal to the number of character positions in the data item referenced by the subject of this entry. If the data item referenced by data-name-2 is specified with a level-number of 01 and it is not declared to be an external data record, there is no such constraint.

7. Data-name-2 must not be qualified even if it is not unique, since no ambiguity of reference exists in this case because of the required placement of the REDEFINES clause within the source program.

8. Multiple re-definitions of the same character positions are permitted. Multiple re-definitions of the same character positions must all use the data-name of the entry that originally defined the area. However data-name-2 may be subordinate to an entry whose Data Description contains a REDEFINES clause.

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

10. No entry having a level-number numerically lower than the level-number of data-name-2 and the subject of the entry may occur between the Data Description entries of data-name-2 and the subject of the entry.

11. The entries giving the new description of the character positions must follow the entries defining the area of data-name-2 without intervening entries that define new character positions.

|12. The REDEFINES clause must not be specified in a Data Description entry that defines an elementary data item whose usage is implicitly or explicitly BIT.

13. Data-name-2 must not reference an elementary data item whose usage is implicitly or explicitly BIT.|

General Rules

1. Storage allocation starts at data-name-2 and continues over a storage area sufficient to contain the number of character positions in the data item referenced by the data-name-1 or FILLER clause.

2. When the same character position is defined by more than one Data Description entry, the data-name associated with any of those Data Description entries can be used to reference that character position.

9.23 RENAMES

Description

The RENAMES clause permits alternative, possibly overlapping, groupings of elementary items.

Format

```
                                          {THROUGH}
66  data-name-1 RENAMES data-name-2 [{       } data-name-3]
                                          {THRU   }
```

Note: Level-number 66 and data-name-1 are shown in the above format to improve clarity. Level-number and data-name-1 are not part of the RENAMES clause.

Syntax Rules

1. Any number of RENAMES entries may be written for a logical record.

2. All RENAMES entries referring to data items within a given logical record must immediately follow the last Data Description entry of the associated Record Description entry.

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

4. Data-name-2 and data-name-3 must be names of elementary items or groups of elementary items in the same logical record, and cannot be the same data-name. A 66 level entry cannot rename another 66 level entry nor can it rename a 01, 77 or 88 level entry.

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

6. None of the items within the range including data-name-2 and data-name-3, if specified, can be variable occurrence data items.

7. The words THRU and THROUGH are equivalent.

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

|9. If the THROUGH phrase is specified, neither data-name-2 nor data-name-3 may reference an elementary data item whose usage is implicitly or explicitly BIT.|

General Rules

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

2. When data-name-3 is not specified, all the data attributes of data-name-2 become the data attributes for data-name-1.

9.24 REPORT

Description

The REPORT clause specifies the names of reports that comprise a report file.

Format

```
{REPORT IS  }
{           } {report-name-1}...
{REPORTS ARE}
```

Syntax Rules

1. Each report-name specified in a REPORT clause must be the subject of a Report Description entry in the Report Section of the same program. The order of appearance of the report-names is not significant.

2. A report-name must appear in only one REPORT clause.

3. The subject of a File Description entry that specifies a REPORT clause may be referred in the Procedure Division only by the USE statement, the CLOSE statement or the OPEN statement with the OUTPUT or EXTEND phrase.

General Rules

1. The presence of more than one report-name in a REPORT clause indicates that the file contains more than one report.

2. After execution of an INITIATE statement and before the execution of a TERMINATE statement for the same report file, the report file is under the control of the RWCS. While a report file is under the control of the RWCS no input-output statement may be executed which references that report file.

3. If the associated file connector is an external file connector, in the run unit, every File Description entry which is associated with that file connector must describe it as a report file.

9.25 SIGN

Description

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

Format

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

Syntax Rules

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

2. Numeric Data Description entries to which the SIGN clause applies must be described implicitly or explicitly as USAGE IS DISPLAY.

3. If the CODE-SET clause is specified in a File Description entry, any signed numeric Data Description entries associated with that File Description entry must be described with the SIGN IS SEPARATE clause.

4. If the SIGN clause is included in a Report Group Description entry, the SEPARATE CHARACTER phrase must be specified.

|5. The SIGN clause may not be specified for a floating-point numeric data item.|

General Rules

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

2. If a SIGN clause is specified in a group item subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in the subordinate group takes precedence for that subordinate group item.

3. If a SIGN clause is specified in an elementary numeric Data Description entry subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in the subordinate elementary numeric Data Description entry takes precedence for that elementary numeric data item.

4. A fixed-point 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, the SIGN IS TRAILING clause (without the SEPARATE phrase) is assumed, |unless the DISPLAY SIGN clause of the Control Division specifies otherwise (see Chapter 5).|

   General rules 5 through 7 do not apply to such signed numeric data items.

5. 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 fixed-point 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).

6. 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 fixed-point 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.

7. 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.

9.26 SOURCE

Description

The SOURCE clause identifies the sending data item that is moved to an associated printable item defined within a Report Group Description entry.

Format

```
SOURCE IS identifier-1
```

Syntax Rules

1. Identifier-1 may be defined in any section of the Data Division. If identifier-1 is a Report Section item it can only be:

   a. PAGE-COUNTER, or

   b. LINE-COUNTER, or

   c. Sum counter that is part of the report within which the SOURCE clause appears.

2. Identifier-1 specifies the sending data item of the implicit MOVE statement that the RWCS will execute to move the contents of the data item referenced by identifier-1 to the printable item. Identifier-1 must be defined such that it conforms to the rules for sending items in the MOVE statement. (See the "MOVE Statement", Chapter 12).

General Rule

The RWCS formats the print lines of a report group just prior to presenting the report group. (See the "TYPE Clause", this chapter). It is at this time that the implicit MOVE statements specified by SOURCE clauses are executed by the RWCS.

9.27 SUM

Description

The SUM clause establishes a sum counter and names the data items to be summed.

Format

```
{SUM {identifier-1}... [UPON {data-name-1}... ]}...

                {data-name-2}
     [RESET ON  {           }]
                {FINAL      }
```

Syntax Rules

1. The data item that is the subject of the Report Group Description entry in which the SUM clause appears must not be defined as alphabetic |or boolean.| Identifier-1 must reference a numeric data item. If identifier-1 is defined in the Report Section, identifier-1 must reference a sum counter.

   If the UPON phrase is omitted, any identifiers in the associated SUM clause which are themselves sum counters must be defined either in the same report group that contains this SUM clause or in a report group which is at a lower level in the control hierarchy of this report.

   If the UPON phrase is specified, any identifiers in the associated SUM clause must not be sum counters.

2. Data-name-1 must be the name of a DETAIL report group described in the same report as the CONTROL FOOTING report group in which the SUM clause appears. Data-name-1 may be qualified by a report-name.

3. A SUM clause can appear only in the description of a CONTROL FOOTING report group.

4. Data-name-2 must be one of the data-names specified in the CONTROL clause for this report. Data-name-2 must not be a lower level control than the associated control for the report group in which the RESET phrase appears.

   FINAL, if specified in the RESET phrase, must also appear in the CONTROL clause for this report.

5. The highest permissible qualifier of a sum counter is the report-name.

General Rules

1. The SUM clause establishes a sum counter. The sum counter is a compiler-generated numeric data item with an operational sign. The size and decimal point location of the sum counter depend on the category of the data item specified by the Report Group Description entry in which the SUM clause is defined. They are determined as follows:

   a. If the associated data item is numeric, the size and decimal point location of the sum counter are the same as those of that data item.

   b. If the associated data item is numeric edited, the size of the sum counter is the number of digit positions of that data item, and the decimal point location is the same as that of the associated data item.

   c. If the associated data item is alphanumeric or alphanumeric edited, the size of the sum counter is the size of this data item, excluding any editing characters, or 18 decimal digits |(30 decimal digits if the compilation is run with LEVEL=NSTD),| whichever is smaller, and the sum counter is an integer.

2. At object time, the Report Writer Control System (RWCS) adds into the sum counter the value contained in each data referenced by identifier-1. This addition is consistent with the rules for arithmetic statements (see "Arithmetic Statements" and "Overlapping Operands", Chapter 10).

3. Only one sum counter exists for an elementary report entry regardless of the number of SUM clauses specified in the elementary report entry.

4. If the elementary report entry for a printable item contains a SUM clause, the sum counter serves as a source data item. The RWCS moves the data contained in the sum counter, according to the rules of the MOVE statement, to the printable item for presentation.

5. If a data-name appears as the subject of an elementary report entry that contains a SUM clause, the data-name is the name of the sum counter; the data-name is not the name of the printable item that the entry may also define.

6. It is permissible for Procedure Division statements to alter the contents of sum counters.

7. Addition of the values of the data items referenced by identifiers into sum counters is performed by the RWCS during the execution of GENERATE and TERMINATE statements. There are three categories of sum counter incrementing called subtotalling, cross footing, and rolling forward. Subtotalling is accomplished during execution of GENERATE statements only, after any control break processing but before processing of the DETAIL report group. (See the "GENERATE Statement", Chapter 11). Cross footing and rolling forward are accomplished during the processing of CONTROL FOOTING report groups. (See the "TYPE Clause", this chapter).

8. The UPON phrase provides the capability to accomplish selective subtotalling for the DETAIL report groups named in the phrase.

9. The RWCS adds each individual addend into the sum counter at a time that depends upon the characteristics of the addend.

   a. When the addend is a sum counter defined in the same CONTROL FOOTING report group, then the accumulation of that addend into the sum counter is termed cross footing.

      Cross footing occurs when a control break takes place and at the time the CONTROL FOOTING report group is processed.

      Cross footing is performed according to the sequence in which sum counters are defined within the CONTROL FOOTING report group. That is, all cross footing into the first sum counter defined in the CONTROL FOOTING report group is completed, and then all cross footing into the second sum counter defined in the CONTROL FOOTING report group is completed. This procedure is repeated until all cross footing operations are completed.

      When one of the addends is the sum counter defined by the Data Description entry in which that SUM clause appears, the initial value of that sum counter at the time of summation is used in the summing operation.

   b. When the addend is a sum counter defined in a lower level CONTROL FOOTING report group, then the accumulation of that addend into the sum counter is termed rolling forward. A sum counter in a lower level CONTROL FOOTING report group is rolled forward when a control break occurs and at the time that the lower level CONTROL FOOTING report group is processed.

   c. When the addend is not a sum counter the accumulation into a sum counter of such an addend is called subtotalling. If the SUM clause contains the UPON phrase, the addends are subtotalled when a GENERATE statement for the designated DETAIL report group is executed. If the SUM clause does not contain the UPON phrase, the addends which are not sum counters are subtotalled when any GENERATE data-name statement is executed for the report in which the SUM clause appears.

10. If two or more of the identifiers specify the same addend, then the addend is added into the sum counters as many times as the addend is referenced in the SUM clause. It is permissible for two or more of the data-names to specify the same DETAIL report group. When a GENERATE data-name statement for such a DETAIL report group is given, the incrementing occurs repeatedly, as many times as data-name appears in the UPON phrase.

11. The subtotalling that occurs when a GENERATE report-name statement is executed is discussed in the appropriate paragraph (see the "GENERATE Statement", Chapter 11).

12. In the absence of an explicit RESET phrase, the RWCS will set a sum counter to zero at the time that the RWCS is processing the CONTROL FOOTING report group within which the sum counter is defined. If an explicit RESET phrase is specified, then the RWCS will set the sum counter to zero at the time that the RWCS is processing the designated level of the control hierarchy. (See the "TYPE Clause", this chapter).

    Sum counters are initially set to zero by the RWCS during the execution of the INITIATE statement for the report containing the sum counter.

9.28 SYNCHRONIZED

Description

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

Format

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

Syntax Rules

1. This clause can appear only with an elementary item.

2. SYNC is an abbreviation for SYNCHRONIZED.

3. SYNCHRONIZED not followed by either RIGHT or LEFT is equivalent to SYNCHRONIZED LEFT.

General Rules

1. 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 the rightmost 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 number of character positions allocated when any such group item is the object of a REDEFINES clause. The unused character positions are not included in the character positions re-defined when the elementary item is the object of a REDEFINES clause.

2. SYNCHRONIZED 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.

3. 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.

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

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

6. If the Data Description of an item contains an operational sign and any form of the SYNCHRONIZED clause, the sign of the item appears in this sign position explicitly or implicitly specified by the SIGN clause.

7. When the SYNCHRONIZED clause is specified in the 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 for each occurrence of these data items (see "General Rule" 3b).

8. Data Allocation in Chapter 3 specifies how elementary items associated with this clause are handled regarding:

    a. The format of records groups containing elementary items whose data description contains the SYNCHRONIZED clause.

    b. Any necessary generation of implicit FILLER, if the elementary item immediately preceding an item containing the SYNCHRONIZED clause does not terminate at an appropriate natural boundary. Such automatically generated FILLER positions are included in

        (i) The size of any group to which the FILLER item belongs, and

        (ii) The number of character positions allocated when the group item of which the FILLER item is part appears as the object of a REDEFINES clause.

9. Alignment boundaries depend upon the USAGE specified for the elementary items, as shown below:

USAGE	Alignment Boundaries
BINARY	Half-word or word
|BIT	Byte|
COMPUTATIONAL	Byte
|COMPUTATIONAL-1	Half word or word
COMPUTATIONAL-2	word
COMPUTATIONAL-3	Byte
COMPUTATIONAL-5	Byte
COMPUTATIONAL-8	Byte
COMPUTATIONAL-9	Byte
COMPUTATIONAL-10	Double-word
COMPUTATIONAL-15	Double-word|
DISPLAY	Byte
INDEX	Byte
PACKED DECIMAL	Byte
|POINTER	Word|

10. The rules for synchronization of the records, as this effects the synchronization of elementary items, are specified in Data Allocation, Chapter 3.

9.29 TYPE

Description

The TYPE clause specifies the particular type of report group that is described by this entry and indicates the time at which the report group is to be processed by the Report Writer Control System.

Format

```
          {{REPORT HEADING}                 }
          {{RH            }                 }
          {                                 }
          {{PAGE HEADING}                   }
          {{PH          }                   }
          {                                 }
          {{CONTROL HEADING} {data-name-1}}
          {{CH              } {FINAL      }}
          {                                 }
TYPE IS   {{DETAIL}                         }
          {{DE    }                         }
          {                                 }
          {{CONTROL FOOTING} {data-name-2}}
          {{CF              } {FINAL      }}
          {                                 }
          {{PAGE FOOTING}                   }
          {{PF          }                   }
          {                                 }
          {{REPORT FOOTING}                 }
          {{RF            }                 }
```

Syntax Rules

1. RH is an abbreviation for REPORT HEADING.

   PH is an abbreviation for PAGE HEADING.

   CH is an abbreviation for CONTROL HEADING.

   DE is an abbreviation for DETAIL.

   CF is an abbreviation for CONTROL FOOTING.

   PF is an abbreviation for PAGE FOOTING.

   RF is an abbreviation for REPORT FOOTING.

2. Report groups specified by REPORT HEADING, PAGE HEADING, CONTROL HEADING FINAL, CONTROL FOOTING FINAL, PAGE FOOTING, and REPORT FOOTING may each appear no more than once in the description of a report.

3. PAGE HEADING and PAGE FOOTING report groups may be specified only if a PAGE clause is specified in the corresponding Report Description entry.

4. Data-name-1, data-name-2 and FINAL, if present, must be specified in the CONTROL clause of the corresponding Report Description entry. At most, one CONTROL HEADING report group and one CONTROL FOOTING report group can be specified for each data-name or FINAL in the CONTROL clause of the Report Description entry. However, neither a CONTROL HEADING report group nor a CONTROL FOOTING report group is required for a data-name or FINAL specified in the CONTROL clause of the Report Description entry.

5. In CONTROL FOOTING, PAGE HEADING, PAGE FOOTING, and REPORT FOOTING report groups, SOURCE clauses and associated USE statements must not reference any of the following:

   a. Group data items containing a control data item.

   b. Data items subordinate to a control data item.

   c. A re-definition or renaming of any part of a control data item.

   In PAGE HEADING and PAGE FOOTING report groups, SOURCE clauses and USE statements must not reference control data-names.

6. When a GENERATE report-name statement is specified in the Procedure Division, the corresponding Report Description entry must include no more than one DETAIL report group. If no GENERATE data-name statements are specified for such a report, a DETAIL report group is not required.

7. The description of a report must include at least one body group.

General Rules

1. DETAIL report groups are processed by the RWCS as a direct result of GENERATE statements. If a report group is other than TYPE DETAIL, its processing is an automatic RWCS function.

2. The REPORT HEADING phrase specifies a report group that is processed by the RWCS only once, per report, as the first report group of that report. The REPORT HEADING report group is processed during the execution of the chronologically first GENERATE statement for that report.

3. The PAGE HEADING phrase specifies a report group that is processed by the RWCS as the first report group on each page of that report except under the following conditions:

   a. A PAGE HEADING report group is not processed on a page that is to contain only a REPORT HEADING report group or only a REPORT FOOTING report group.

   b. A PAGE HEADING report group is processed as the second report group on a page when it is preceded by a REPORT HEADING report group that is not to be presented on a page by itself. (See "Presentation Rules Tables", Chapter 8).

4. The CONTROL HEADING phrase specifies a report group that is processed by the RWCS at the beginning of a control group for a designated control data-name or, in the case of FINAL, is processed during the execution of the chronologically first GENERATE statement for that report. During the execution of any GENERATE statement at which the RWCS detects a control break, any CONTROL HEADING report groups associated with the highest control level of the break and lower levels are processed.

5. The DETAIL phrase specifies a report group that is processed by the RWCS when a corresponding GENERATE statement is executed.

6. The CONTROL FOOTING phrase specifies a report group that is processed by the RWCS at the end of a control group for a designated control data-name.

   In the case of FINAL, the CONTROL FOOTING report group is processed only once per report as the last body group of that report. During the execution of any GENERATE statement in which the RWCS detects a control break, any CONTROL FOOTING report group associated with the highest level of the control break or more minor levels is presented. All CONTROL FOOTING report groups are presented during the execution of the TERMINATE statement if there has been at least one GENERATE statement executed for the report. (See the "TERMINATE Statement", Chapter 13).

7. The PAGE FOOTING phrase specifies a report group that is processed by the RWCS as the last report group on each page except under the following conditions:

   a. A PAGE FOOTING report group is not processed on a page that is to contain only a REPORT HEADING report group or only a REPORT FOOTING report group.

   b. A PAGE FOOTING report group is processed as the second to last report group on a page when it is followed by a REPORT FOOTING report group that is not to be processed on a page by itself (see "Presentation Rules Tables", Chapter 8).

8. The REPORT FOOTING phrase specifies a report group that is processed by the RWCS only once per report and as the last report group of that report. The REPORT FOOTING report group is processed during the execution of a corresponding TERMINATE statement, if there has been at least one GENERATE statement executed for the report. (See the "TERMINATE Statement", Chapter 13).

9. The sequence of steps that the RWCS executes when it processes a REPORT HEADING, PAGE HEADING, CONTROL HEADING, PAGE FOOTING, or REPORT FOOTING report group is described below.

   a. If there is a USE BEFORE REPORTING procedure that references the data-name of the report group, the USE procedure is executed.

   b. If a SUPPRESS statement has been executed or if the report group is not printable, there is no further processing to be done for the report group.

   c. If a SUPPRESS statement has not been executed and the report group is printable, the RWCS formats the print lines and presents the report group according to the presentation rules for that type of report group. (See "Presentation Rules Tables", Chapter 8).

10. The sequence of steps that the RWCS executes when it processes a CONTROL FOOTING report group is described below.

    The GENERATE rules specify that when a control break occurs, the RWCS produces the CONTROL FOOTING report groups beginning at the minor level, and proceeding upwards, through the level at which the highest control break was sensed. In this regard, it should be noted that even though no CONTROL FOOTING report group has been defined for a given control data-name, the RWCS will still have to execute the step described in paragraph 10f below if a RESET phrase within the report description specifies that control data-name.

    a. Sum counters are cross footed, i.e., all sum counters defined in this report group that are operands of SUM clauses in the same report group are added to their sum counters. (See the "SUM Clause", this chapter).

    b. Sum counters are rolled forward, i.e., all sum counters defined in the report group that are operands of SUM clauses in higher level CONTROL FOOTING report groups are added to the higher level sum counters. (See the "SUM Clause", this chapter).

    c. If there is a USE BEFORE REPORTING procedure that references the data-name of the report group the USE procedure is executed.

    d. If a SUPPRESS statement has been executed or if the report group is not printable, the RWCS next executes the step described in paragraph 10f below.

    e. If a SUPPRESS statement has not been executed and the report group is printable, the RWCS formats the print lines and presents the report group according to the presentation rules for CONTROL FOOTING report groups.

    f. Then the RWCS resets those sum counters that are to be reset when the RWCS processes this level in the control hierarchy. (See the "SUM Clause", this chapter).

11. The DETAIL report group processing that the RWCS executes in response to a GENERATE data-name statement is described in paragraphs 11a through 11e below.

    When the description of a report includes exactly one DETAIL report group, the detail-related processing that the RWCS executes in response to a GENERATE report-name statement is described in paragraphs 11a through 11e below. These steps are performed as though a GENERATE data-name statement were being executed.

    When the description of a report includes no detail report groups, the detail-related processing that the Report Writer Control System executes in response to a GENERATE report-name statement is described in paragraph 11a. This step is performed as though the description of the report included exactly one DETAIL report group, and a GENERATE data-name statement were being executed.

    a. The RWCS performs any subtotalling that has been designated for the DETAIL report group. (See the "SUM Clause", this chapter).

    b. If there is a USE BEFORE REPORTING procedure that refers to the data-name of the report group, the USE procedure is executed.

    c. If a SUPPRESS statement has been executed or if the report group is not printable there is no further processing done for the report group.

d. If the DETAIL report group is being processed as a consequence of a GENERATE report-name statement, there is no further processing done for the report group.

e. If neither 11c nor 11d above applies, the RWCS formats the print lines and presents the report group according to the presentation rules for DETAIL report groups. (See "Presentation Rules Tables", Chapter 8).

12. When the RWCS is processing a CONTROL HEADING, CONTROL FOOTING, or DETAIL report group, as described in General Rules 9, 10, and 11, the RWCS may have to interrupt the processing of that body group after determining that the body group is to be presented, and execute a page advance (and process PAGE FOOTING and PAGE HEADING report groups) before actually presenting the body group.

13. During control break processing, the values of control data items that the RWCS used to detect a given control break are referred to as prior values.

    a. During control break processing of a CONTROL FOOTING report group, any references to control data items in a USE procedure or SOURCE clause associated with that CONTROL FOOTING report group are supplied with prior values.

    b. When a TERMINATE statement is executed, the RWCS makes the prior control data item values available to SOURCE clause or USE procedure references in CONTROL FOOTING and REPORT FOOTING report groups as though a control break had been detected in the highest control data-name.

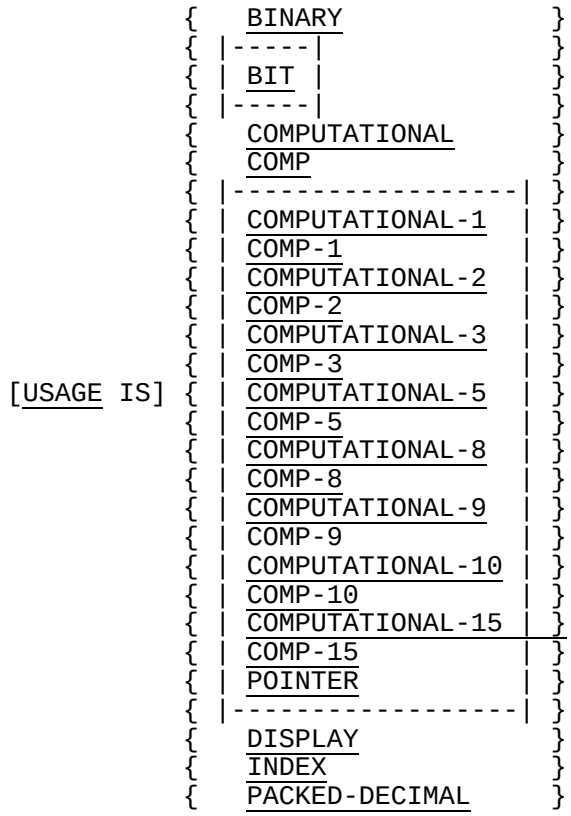
    c. All other data item references within report groups and their USE procedures access the current values that are contained within the data items at the time the report group is processed.

9.30 USAGE

Description

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

Format



Syntax Rules

1. The USAGE clause may be written in any Data Description entry with a level-number other than 66 or 88.

2. If the USAGE clause is written in the Data Description entry for a group item, it may also be written in the Data Description entry for any subordinate elementary item or group item, but the same usage must be specified in both entries.

|3. Unless otherwise specified in the DEFAULT SECTION of the CONTROL DIVISION, COMPUTATIONAL is equivalent to COMPUTATIONAL-3. The use of the COMPUTATIONAL IS phrase of the DEFAULT SECTION of the CONTROL DIVISION allows the user to make COMPUTATIONAL equivalent to COMPUTATIONAL-1, 2, 3, 5 or 8, to BINARY, to PACKED-DECIMAL or to DISPLAY. In the text below, the rules applicable to COMPUTATIONAL-1, 2, 3, 5 or 8, to BINARY, to PACKED-DECIMAL or to DISPLAY are also applicable to COMPUTATIONAL if COMPUTATIONAL is, or has been made, equivalent to the relevant USAGE.|

4. An elementary data item whose declaration contains, or an elementary data item subordinate to a group data item whose declaration contains a USAGE clause specifying BINARY, COMPUTATIONAL, |COMPUTATIONAL-3, -5 or -8| or PACKED-DECIMAL must be declared with a PICTURE character-string that defines a fixed-point numeric item, i.e. a PICTURE character-string that contains only the symbols 'P', 'S', 'V' and '9' (see the "PICTURE Clause", this chapter).

   If the specified usage is BINARY, the PICTURE character-string must not contain more than 18 digits.

|5. An elementary data item whose declaration contains, or an elementary data item subordinate to a group item whose declaration contains a USAGE clause with the BIT phrase must be declared with a PICTURE character-string that describes a boolean data item, i.e. a PICTURE character-string that contains only the symbol '1' (see the "PICTURE Clause", this chapter).

6. The PICTURE clause is not mandatory for data whose USAGE IS COMPUTATIONAL-1 or 2. When absent, a PICTURE S9(4) or S9(9) is assumed respectively. When present the PICTURE clause must specify an integer without scaling position. The sign is always considered to be present, even if the PICTURE character string does not show it. When the USAGE IS COMPUTATIONAL-1, and the PICTURE clause shows more than 4 digit positions the item is allocated, synchronized and handled as if it was a COMPUTATIONAL-2 item.

7. An entry describing a COMPUTATIONAL-9 ,-10 or -15 item must not contain a PICTURE clause. The ALL literal figurative constant must not be used in a VALUE clause, a MOVE statement, or a condition involving a COMPUTATIONAL-9 ,-10 or -15 item.|

8. COMP is an abbreviation for COMPUTATIONAL.

   |COMP-n is an abbreviation for COMPUTATIONAL-n.|

9. The USAGE clause for a report group item can specify only USAGE IS DISPLAY.

10. A usage other than DISPLAY must not be specified for a data item whose Data Description entry contains the BLANK WHEN ZERO, JUSTIFIED or SIGN clause.

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

12. PICTURE or VALUE clauses cannot be used for data items whose usage is INDEX.

13. An elementary data item described with a USAGE IS INDEX clause must not be a conditional variable.

|14. A pointer 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.

15. The PICTURE clause cannot be used for data items whose usage is pointer.

16. DPS8 usages are accepted and interpreted as the closest DPS 7 usage, namely: COMP-6, COMP-7, COMP-11, COMP-12, COMP-13 and COMP-14 become COMP-2, COMP-1, COMP-9, COMP-10 COMP-9 and COMP-10 respectively.|

General Rules

1. The usage of a group item is always implicitly DISPLAY. If the USAGE clause is written at a group level, it applies to each elementary item in the group, but not to the group item itself.

2. The USAGE clause specifies the manner in which a data item is represented in the storage medium of the 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 some operands referred to. The USAGE clause may affect the radix or type of character representation of the item.

3. The USAGE IS BINARY clause specifies that a radix of two (2) is used to represent a numeric item in the storage of the computer.

|4. The USAGE IS BIT clause specifies that bits are used to represent a boolean data item. The alignment of an elementary boolean data item whose usage is either explicitly or implicitly BIT is determined by the following rules:

   a. If the current data item is the first elementary data item in a group, the current data item is aligned on the same character boundary as the group item.

   b. If the current data item is the first data item following the last data item of a group, the current data item is aligned on a character boundary.

   c. If the preceding data item does not have a USAGE IS BIT clause implicitly or explicitly stated, the current data item is aligned on a character boundary.

   d. If the preceding data item has a USAGE IS BIT clause implicitly or explicitly stated, and a SYNCHRONIZED clause, the current data item is aligned on a character boundary.

   e. If the preceding data item has a USAGE IS BIT clause and none of the above rules apply, the current data item is aligned on the bit following the preceding data item.

   This alignment may cause the generation of implicit FILLER positions.|

5. The USAGE IS COMPUTATIONAL, |COMPUTATIONAL-1, COMPUTATIONAL-3, COMPUTATIONAL-5, COMPUTATIONAL-8, COMPUTATIONAL-9, COMPUTATIONAL-10 or COMPUTATIONAL-15| clause is used to represent a numeric item in the storage of the computer. If a group item is described with such a USAGE, the elementary items in the group have this USAGE. The group item itself does not have it (cannot be used in computations).

6. The USAGE IS DISPLAY clause (whether specified explicitly or implicitly) specifies that a Standard Data Format is used to represent a data item in the storage of the computer (one character stored in one byte coded in EBCDIC), and that the data item is aligned on a character boundary.

7. 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.

8. When a MOVE statement or an input-output statement that references a group data item which contains a data item whose usage is not DISPLAY is executed, no conversion of that data item takes place.

9. The USAGE IS INDEX clause specifies that a data item is an index data item and contains a value which must correspond to an occurrence number of a table element.

10. The USAGE IS PACKED-DECIMAL clause specifies that a radix of 10 is used to represent a numeric item in the storage of the computer. Furthermore, this clause specifies that each digit position must occupy the minimum possible configuration in computer storage.

|11. The USAGE IS POINTER clause specifies that a data item is a pointer data item and contains a value which must correspond to the address of a data item.|

12. The following representation is given to data depending on their USAGE.

    BINARY

    PICTURE clause shows:

less than 5 digit positions	16-bit signed binary data with the leftmost bit showing the sign
5 to 9 digit positions	32-bit signed binary data with the leftmost bit showing the sign
10 to 14 digit positions	48-bit signed binary data with the leftmost bit showing the sign
15 to 18 digit positions	64-bit signed binary data with the leftmost bit showing the sign
	In all cases, if the item is unsigned and is a receiving item, the sign is '+', irrespective of the sign of the actual result

COMPUTATIONAL

|COMPUTATIONAL-1

PICTURE clause shows

less than 5 digit positions	16-bit signed binary data
more than 4 digit positions	32-bit signed binary data
PICTURE clause not present	16-bit signed binary data
COMPUTATIONAL-2	32-bit signed binary data

COMPUTATIONAL-3 | (COMPUTATIONAL by default)

PICTURE clause shows:

a sign	packed decimal data with the 4 rightmost bits showing the sign
no sign	packed decimal data with the 4 rightmost bits showing the rightmost digit in both cases, if unused, the leftmost digit must be zero
|COMPUTATIONAL-5	same as COMPUTATIONAL-3 except that, when signed, the sign has the ASCII representation i.e.: hexadecimal 'A', 'B', 'C', 'E' and 'F' for '+' ('B' is the standard representation for '+') and hexadecimal 'D' for '-'
COMPUTATIONAL-8	same as PACKED-DECIMAL
COMPUTATIONAL-9	floating single precision 32 bit binary
COMPUTATIONAL-10	floating double precision 64 bit binary
COMPUTATIONAL-15	floating quadruple precision 128 bit binary|
INDEX	32-bit signed binary displacement followed by 16 bit signed binary occurrence number.

PACKED-DECIMAL

PICTURE clause shows:

a sign	packed decimal data with the 4 rightmost bits showing the sign
no sign	packed decimal data with the 4 rightmost bits showing a sign; if the item is a receiving item, the sign is "+", irrespective of the sign of the actual result

|POINTER	32 bit direct ITS	

9.31 VALUE

Description

The VALUE clause defines the value of Report Section printable items, the initial value of Communication Section, Working-Storage Section and Constant Section data items, and the values associated with condition-names.

Format 1

```
VALUE IS literal
```

Format 2

```
{VALUE IS  }                {THROUGH}
{          }   {literal-1 [{       } literal-2]}...
{VALUES ARE}                {THRU   }
```

```
      |-----------------------------------|
      | [WHEN SET TO FALSE IS literal-3] |
      |-----------------------------------|
```

Format 3

```
|---------------|
| VALUE IS NULL |
|---------------|
```

Syntax Rules

All Formats

1. |The VALUE clause may be specified in any entry which is part of the description of an external data record.| The VALUE clause may be specified for condition-names entries associated with such Data Description entries.

Formats 1 and 2

|2. The VALUE clause may be used with a variable length data item. In this case, the initial value of the data item will be determined as if the data item were not described as variable length, i.e., as if its PICTURE character-string did not contain the symbol 'L'.|

3. A signed numeric literal must have associated with it a signed numeric PICTURE character-string |or usage COMP-1, COMP-2, COMP-9, COMP-10 or COMP-15.|

4. All numeric literals in a VALUE clause of an item must have a value which is within the range of values indicated by the PICTURE clause |or implied by its USAGE clause,| and must not have a value that would require truncation of non-zero digits. Non-numeric literals in a VALUE clause of an item must not exceed the size indicated by the PICTURE clause.

Format 2

5. The words THRU and THROUGH are equivalent.

6. Format 2 must be used only in connection with a condition-name.

|7. If the conditional variable associated with the condition-name is a boolean data item, the THROUGH phrase must not be specified.

8. Literal-3 must not be equal to any literal-1, and in any literal-1 through literal-2 pair, if literal-3 is greater than literal-1, it must not be less than or equal to literal-2.

Format 3

9. A format 3 VALUE clause must have an elementary pointer data item associated with it.|

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 or |Constant| Section item, the literal is aligned in the data item according to standard alignment rules (See "Standard Rules for Data Alignment", Chapter 3).

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

   |c. If the category of the item is boolean, all literals in the VALUE clause must be boolean literals. Boolean literals are aligned in the data item according to the standard alignment rules (See "Standard Alignment Rules", Chapter 3).

   d. If the data item is a pointer data item, only the VALUE IS NULL clause is allowed.|

   e. Initialization is not affected by any BLANK WHEN ZERO or JUSTIFIED clause that may be specified.

Condition-name Rules

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

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

|4. The FALSE phrase only has meaning if the associated condition-name is referenced in a SET condition-name TO FALSE statement (See the "SET Statement", Chapter 13).

5. Format 3 may be used in connection with a condition-name whose conditional variable is a pointer data item.|

Data Description Entries Other Than Condition-names

6. Rules governing the use of the VALUE clause differ in the respective sections of the Data Division:

   a. In the File and Linkage Sections, the VALUE clause may be used only in condition-name entries. Therefore the initial value of the data-item of the File Section is undefined.

   b. In the Working-Storage |, Constant| and Communication Sections, the VALUE clause must be used in condition-name entries. VALUE clauses in the Working-Storage, |Constant| and Communication Sections of a program take effect only when the program is placed into its initial state. If the VALUE clause is used in the description of the data item, the data item is initialized to the defined value. If the VALUE clause is not associated with a data item, the initial contents of the data item are undefined.

   c. In the Report Section, if the elementary report entry containing the VALUE clause does not contain a GROUP INDICATE clause, then the printable item will assume the specified value each time its report group is printed. However, when the GROUP INDICATE clause is also present, the specified value will be presented only when certain object time conditions exist (see the "GROUP INDICATE Clause", this chapter).

7. 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. The rule does not apply to condition-name entries.

8. If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a non-numeric 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.

9. The VALUE clause must not be specified for a group containing items subordinate to it with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).

10. If a VALUE clause is specified in a Data Description entry of a data item which is associated with a variable occurrence data item, the initialization of the data item behaves as if the value of the data item reference by the DEPENDING ON phrase in the OCCURS clause specified for the variable occurrence data item is set to the maximum number of occurrences as specified by that OCCURS clause. A data item is associated with a variable occurrence data item in any of the following cases:

    a. It is a group data item which contains a variable occurrence data item.

    b. It is a variable occurrence data item.

    c. It is a data item which is subordinate to a variable occurrence data item.

    If a VALUE clause is associated with the data item referenced by a DEPENDING ON phrase, that value is considered to be placed in the data item after the variable occurrence data item is initialized (See the "OCCURS Clause", this chapter).

11. A format 1 VALUE clause specified in a Data Description entry that contains an OCCURS clause or in an entry that is subordinate to an OCCURS clause causes every occurrence of the associated data item to be assigned the specified value.

9.32 VALUE OF

Description

The VALUE OF clause particularizes the description of an item in the label records associated with a file. The VALUE OF clause is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

Format

```
                      {data-name-1}
VALUE OF {name-1 IS   {           }}
                      {literal-1  }
```

Syntax Rules

1. Data-name-1 should be qualified when necessary, but cannot be subscripted nor can data-name-1 be an item described with the USAGE IS INDEX clause.

2. Data-name-1 must be in the Working-Storage |or the Constant| Section.

3. Name-1 must obey the rules for the formation of a COBOL word.

General Rule

1. This clause is given for documentation only. It is accepted for compatibility.

10. Procedure Division - Overview

This chapter introduces the Procedure Division.

10.1 GENERAL DESCRIPTION

10.1.1 The Procedure Division 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 the "USE Statement", Chapter 13.)

10.1.2 Procedures

A procedure is composed of a paragraph, or group of successive paragraphs, or a section, or a group of successive sections within the Procedure Division. If one paragraph is in a section, 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.

A sentence consists of one or more statements and is terminated by the separator period.

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.

10.1.3 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.

10.1.4 Procedure Division Structure

10.1.4.1 Procedure Division Header

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

```
PROCEDURE DIVISION [USING {data-name-1}...].
```

The USING phrase is necessary only if the object program is to be invoked by a CALL statement and that statement includes a USING phrase.

The USING phrase of the Procedure Division header identifies the names used by the program for any parameters passed to it by a calling program. The parameters passed to a called program are identified in the USING phrase of the calling program's CALL statement. The correspondence between the two lists of names is established on a positional basis. The data description of each parameter in the CALL statement must be the same as the data description of the corresponding parameter in the USING phrase of the Procedure Division header. (See the "CALL Statement", Chapter 11.)

Data-name-1 must be defined as a level 01 entry or a level 77 entry in the Linkage Section. A particular user-defined word may not appear more than once as data-name-1. Data-name-1 must not be the name of a data item possessing the external attribute. The Record Description entry for data-name-1 must not contain a REDEFINES clause. Data-name-1 may, however, be the object of a REDEFINES clause elsewhere in the Linkage Section.

The following additional rules apply:

1. If the reference to the corresponding data item in the CALL statement declares the parameter to be passed by content, the value of the item is moved when the CALL statement is executed and placed into a system-defined storage item possessing the attributes declared in the Linkage Section for data-name-1.

   |If the called program is not in the same separately compiled program as the calling program,| the data description of each parameter in the BY CONTENT phrase of the CALL statement must be the same, meaning no conversion or extension or truncation, as the data description of the corresponding parameter in the USING phrase of the Procedure Division header. (See the "CALL Statement", Chapter 11.)

2. If the reference to the corresponding data item in the CALL statement declares the parameter to be passed by reference, the object program operates as if the data item in the called program occupies the same storage area as the data item in the calling program. The description of the data item in the called program must describe the same number of character positions as described by the description of the corresponding data item in the calling program.

3. At all times in the called program, references to data-name-1 are resolved in accordance with the description of the item given in the Linkage Section of the called program.

4. Data items defined in the Linkage Section of the called program may be referenced within the Procedure Division of that program if, and only if, they satisfy one of the following conditions:

   a. They are operands of the USING phrase of the Procedure Division header.

   b. They are subordinate to operands of the USING phrase of the Procedure Division header.

   c. They are defined with a REDEFINES or RENAMES clause, the object of which satisfies the above conditions.

   d. They are items subordinate to any item which satisfies the condition in rule 4c.

   e. They are condition-names or index-names associated with data items that satisfy any of the above four conditions.

10.1.4.2 Procedure Division Body

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

Format 1

```
[DECLARATIVES.
{section-name [GLOBAL] SECTION [segment-number].
USE statement.
[paragraph-name [GLOBAL]. [sentence]...]...}...
END DECLARATIVES.]
{section-name [GLOBAL] SECTION [segment-number].
[paragraph-name [GLOBAL]. [sentence]...]...}...
```

Format 2

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

10.2 STATEMENTS AND SENTENCES

There are four types of statements: imperative statements, conditional statements, compiler directing statements, and delimited scope statements.
There are three types of sentences: imperative sentences, conditional sentences, and compiler directing sentences.

10.2.1 Conditional Statements and Sentences

10.2.1.1 Definition of 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:

1. An EVALUATE, IF, SEARCH or RETURN statement.

2. A READ statement that specifies the AT END, NOT AT END, INVALID KEY, or NOT INVALID KEY phrase.

3. A WRITE statement that specifies the INVALID KEY, NOT INVALID KEY, END-OF-PAGE, or NOT END-OF-PAGE phrase.

4. A START, REWRITE or DELETE statement that specifies the INVALID KEY or NOT INVALID KEY phrase.

5. An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) that specifies the SIZE ERROR or NOT ON SIZE ERROR phrase.

6. A RECEIVE statement that specifies a NO DATA or WITH DATA phrase.

7. A STRING or UNSTRING statement that specifies the ON OVERFLOW or NOT ON OVERFLOW phrase.

8. A CALL statement that specifies the ON OVERFLOW, ON EXCEPTION, |NOT ON OVERFLOW,| or NOT ON EXCEPTION phrase.

10.2.1.2 Definition of Conditional Phrase

A conditional phrase specifies the action to be taken upon determination of the truth value of a condition resulting from the execution of a conditional statement.

A conditional phrase is one of the following:

1. AT END or NOT AT END phrase when specified within a READ statement.
2. INVALID KEY or NOT INVALID KEY phrase when specified within a DELETE, READ, REWRITE, START, or WRITE statement.
3. END-OF-PAGE or NOT END-OF-PAGE phrase when specified within a WRITE statement.
4. SIZE ERROR or NOT ON SIZE ERROR phrase when specified within an ADD, COMPUTE, DIVIDE, MULTIPLY, or SUBTRACT statement.
5. NO DATA or WITH DATA phrase when specified within a RECEIVE statement.
6. ON OVERFLOW or NOT ON OVERFLOW phrase when specified within a STRING or UNSTRING statement.
7. ON OVERFLOW, ON EXCEPTION, |NOT ON OVERFLOW,| or NOT ON EXCEPTION phrase when specified within a CALL statement.

10.2.1.3 Definition of Conditional Sentence

A conditional sentence is a conditional statement, optionally preceded by an imperative statement, terminated by the separator period.

10.2.2 Compiler Directing Statements and Compiler Directing Sentences

10.2.2.1 Definition of Compiler Directing Statement

A compiler directing statement consists of a compiler directing verb and its operands. The compiler directing verbs are COPY, REPLACE and USE (see the "COPY Statement" and the "REPLACE Statement" in Chapter 15, and the "USE Statement" in Chapter 13). A compiler directing statement causes the compiler to take a specific action during compilation.

10.2.2.2 Definition of Compiler Directing Sentence

A compiler directing sentence is a single compiler directing statement terminated by the separator period.

10.2.3 Imperative Statements and Imperative Sentences

10.2.3.1 Definition of Imperative Statement

An imperative statement begins with an imperative verb and specifies an unconditional action to be taken by the object program or is a conditional statement that is delimited by its explicit scope terminator (delimited scope 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	EXIT	RELEASE
ADD (1)	GENERATE	REWRITE (2)
ALTER	GO TO	SEND
CALL (7)	INITIALIZE	SET
CANCEL	INITIATE	SORT
CLOSE	INSPECT	START (2)
COMPUTE (1)	MERGE	STOP
CONTINUE	MOVE	STRING (3)
DELETE (2)	MULTIPLY (1)	SUBTRACT (1)
DISABLE	OPEN	SUPPRESS
DISPLAY	PERFORM	TERMINATE
DIVIDE (1)	PURGE	|TRANSFORM|
ENABLE	READ (5)	UNSTRING (3)
|EXAMINE|	RECEIVE (4)	WRITE (6)

(1) Without the optional ON SIZE ERROR and NOT ON SIZE ERROR phrases

(2) Without the optional INVALID KEY and NOT INVALID KEY phrases

(3) Without the optional ON OVERFLOW and NOT ON OVERFLOW phrases

(4) Without the optional NO DATA and WITH DATA phrases

(5) Without the optional AT END, NOT AT END, INVALID KEY, and NOT INVALID KEY phrases

(6) Without the optional INVALID KEY, NOT INVALID KEY, END-OF-PAGE, and NOT END-OF-PAGE phrases

(7) Without the optional ON EXCEPTION, ON OVERFLOW, NOT ON EXCEPTION, and |NOT ON OVERFLOW| phrases

Whenever '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 by any phrase associated with a statement containing that 'imperative-statement'.

10.2.3.2 Definition of Imperative Sentence

An imperative sentence is an imperative statement terminated by the separator period.

10.2.4 Delimited Scope Statements

Definition

A delimited scope statement is any statement which includes its explicit scope terminator. (See "Explicit and Implicit Scope Terminators", Chapter 3.)

Scope of Statements

Scope terminators delimit the scope of certain Procedure Division statements. Statements which include their explicit scope terminators are termed delimited scope statements. (See "Explicit and Implicit Scope Terminators" in Chapter 3, and "Delimited Scope Statements", this chapter.) The scope of statements which are contained within statements (nested) may also be implicitly terminated.

When statements are nested within other statements, a separator period which terminates the sentence also implicitly terminates all nested statements.

Whenever any statement is contained within another statement, the next phrase of the containing statement following the contained statement terminates the scope of any unterminated contained statement.

When a delimited scope statement is nested within another delimited scope statement with the same verb, each explicit scope terminator terminates the statement begun by the most recently preceding, and as yet unterminated, occurrence of that verb.

When statements are nested within other statements which allow optional conditional phrases, any optional conditional phrase encountered is considered to be the next phrase of the nearest preceding unterminated statement with which that phrase is permitted to be associated according to the general format and the syntax rules for that statement, but with which no such phrase has already been associated. An unterminated statement is one which has not been previously terminated either explicitly or implicitly. (See "Explicit and Implicit Scope Terminators", Chapter 3.)

10.3 ARITHMETIC EXPRESSIONS

10.3.1 Definition of Arithmetic Expression

An arithmetic expression can be:

- An identifier of a numeric elementary item
- A numeric literal
- The figurative constant ZERO (ZEROS, ZEROES)
- Such identifiers, figurative constants and literals, separated by arithmetic operators
- Two arithmetic expressions separated by an arithmetic operator, or
- An arithmetic expression enclosed in parentheses.

Any arithmetic expression may be preceded by a unary operator. The permissible combinations of identifiers, numeric literals, arithmetic operators, and parentheses are given in the table "Combination of Symbols in Arithmetic Expressions" below.

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

10.3.2 Arithmetic Operators

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

Binary Arithmetic Operator	Meaning
+	Addition
-	Subtraction
*****	Multiplication
/	Division
******	Exponentiation

Unary Arithmetic Operator	Meaning
+	The effect of multiplication by the numeric literal +1
-	The effect of multiplication by the numeric literal -1.

10.3.3 Formation and Evaluation Rules

1. Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first, and, within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:

   1st. Unary plus and minus
   2nd. Exponentiation
   3rd. Multiplication and division
   4th. Addition and subtraction

2. Parentheses are used either to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear, or to modify the normal hierarchical sequence of execution in expressions where it is necessary to have some deviation from the normal precedence. When the sequence of execution is not specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right.

3. The ways in which identifiers, literals, operators, and parentheses may be combined in an arithmetic expression are summarized in the table "Combination of Symbols in Arithmetic Expressions" below, where:

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

   b. The letter 'X' indicates an invalid pair.

Table 10-1. Combination of Symbols in Arithmetic Expressions

FIRST SYMBOL	SECOND SYMBOL				
	Identifier or Literal	* / ** - +	Unary + or -	(	)
Identifier or Literal	X	P	X	X	P
* / ** + -	P	X	P	P	X
Unary + or -	P	X	X	P	X
(	P	X	P	P	X
)	X	P	X	X	P

4. An arithmetic expression may only begin with the symbol '(', '+', '-', an identifier, or a literal and may only end with a ')', an identifier, or a literal. There must be a one-to-one correspondence between left and right parentheses of an arithmetic expression such that each left parenthesis is to the left of its corresponding right parenthesis. If the first operator in an arithmetic expression is a unary operator, it must be immediately preceded by a left parenthesis if that arithmetic expression immediately follows an identifier or another arithmetic expression.

5. The following rules apply to evaluation of exponentiation in an arithmetic expression:

   a. If the value of an expression to be raised to a power is zero, the exponent must have a value greater than zero. Otherwise, the size error condition exists (see the "SIZE ERROR Phrase", this chapter).

   b. If the evaluation yields both a positive and negative real number, the value returned as the result is the positive number.

   c. If no real number exists as the result of the evaluation, the size error condition exists.

6. Arithmetic expressions allow the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items. When no reference to a resultant identifier exists in a statement, an intermediate-data-item is used to store the results of arithmetic expressions (see "Intermediate Data Item", this chapter).

10.4 BOOLEAN EXPRESSIONS

10.4.1 Definition of a Boolean Expression

|A boolean expression is an identifier referencing a boolean data item, a boolean literal, the figurative constants ZERO (ZEROS, ZEROES) or ALL literal, where literal is a boolean literal, such identifiers, figurative constants and/or literals separated by a boolean operator, two boolean expressions separated by a boolean operator, or a boolean expression enclosed in parentheses. Any boolean expression may be preceded by the unary boolean operator. The permissible combinations of variables, boolean literals, boolean operators and parentheses are given in the table "Combination of Symbols in Boolean Expressions" below.|

10.4.2 Boolean Operators

|There are three binary boolean operators and one unary boolean operator. They may be used only in boolean expressions. They are represented by the following reserved words that must be both preceded and followed by the separator space.

Binary Boolean Operator:	Meaning:
B-AND	Boolean Conjunction
B-OR	Boolean Inclusive Disjunction
B-EXOR	Boolean Exclusive Disjunction
Unary Boolean Operator:	**Meaning:**
B-NOT	Boolean Negation|

10.4.3 Boolean Formation and Evaluation Rules

|1. Whenever two boolean expressions are separated only by 'B-AND', 'B-OR' or 'B-EXOR', or whenever a boolean expression is immediately preceded by 'B-NOT', the 'B-AND', 'B-OR', 'B-EXOR' or 'B-NOT' is a boolean operator.

2. Parentheses may be used in boolean expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first and, within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:

   1st - Negation (B-NOT)
   2nd - Conjunction (B-AND)
   3rd - Disjunction (B-OR and B-EXOR)

3. Parentheses are used either to eliminate ambiguities in logic, where consecutive operations of the same hierarchical level appear, or to modify the normal hierarchical sequence of execution in expressions where it is necessary to have some deviation from the normal precedence. When the sequence of execution is not specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right.

4. The ways in which identifiers referencing boolean data items, boolean literals, boolean operators, and parentheses may be combined in a boolean expression are summarized in the table "Combinations of Symbols in Boolean Expressions" below.

   In this table:

   a. The letter 'P' indicates a permissible pair.
   b. The character 'X' indicates an invalid pair.

Table 10-2. Combination of Symbols in Boolean Expressions

FIRST SYMBOL	SECOND SYMBOL				
	Identif. or Literal	B-AND B-OR B-EXOR	B-NOT	(	)
Identifier or Literal	X	P	X	X	P
B-AND, B-OR B-EXOR	P	X	P	P	X
B-NOT	P	X	X	P	X
(	P	X	P	P	X
)	X	P	X	X	P

5. A boolean expression may only begin with the symbol '(', an identifier that references a boolean data item, a boolean literal or the operator 'B-NOT'. A boolean expression may only end with the symbol ')', an identifier that references a boolean data item or a boolean literal. There must be a one-to-one correspondence between left and right parentheses of a boolean expression such that each left parenthesis is to the left of its corresponding right parenthesis.

6. Binary boolean operations are performed without regard for the usage of the operands. If the two operands are of equal size, the operation proceeds by conjoining or disjoining boolean characters in corresponding boolean character positions starting from the high order end and continuing to the lower order end. If the operands are of unequal length, then the operation proceeds as though the shorter operand was extended on the low order end by a sufficient number of boolean zeroes to make the operands of equal size.

7. The result of the evaluation of a boolean expression is a hypothetical data item whose size is that of the largest boolean item referenced in the expression.|

10.5 CONDITIONAL EXPRESSIONS

Conditional expressions identify conditions that are tested to enable the object program to select between alternate paths of control depending upon the truth value of the condition. A conditional expression has a truth value represented by either 'true' or 'false'. Conditional expressions are specified in the EVALUATE, IF, PERFORM, and SEARCH statements. There are two categories of conditions associated with conditional expressions: simple conditions and complex conditions. Each may be enclosed within any number of paired parentheses, in which case its category is not changed.

10.5.1 Simple Conditions

The simple conditions are the relation, class, condition-name, switch-status, and sign conditions.

A simple condition has a truth value of 'true' or 'false'. The inclusion in parentheses of simple conditions does not change the simple condition truth value.

10.5.1.1 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, an index-name, the value resulting from an arithmetic |or a boolean| expression. A relation condition has a truth value of 'true' if the relation exists between the operands. Comparison of two numeric operands |or two boolean 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 non-numeric comparison rules apply|; however, if one of the operands is a boolean expression, both operands must be boolean expressions.|

The format for a relation condition |not involving boolean or pointer expressions| is as follows:

```
{identifier-1          }
{literal-1             }
{arithmetic-expression-1}
{index-name-1          }

        {   IS [NOT] GREATER THAN      }
        {   IS [NOT] LESS THAN         }
        {   IS [NOT] EQUAL TO          }
        {   IS GREATER THAN OR EQUAL TO}
        {   IS LESS THAN OR EQUAL TO   }
        {   IS [NOT] >                 } {identifier-2          }
        {   IS [NOT] <                 } {literal-2             }
        {   IS [NOT] =                 } {arithmetic-expression-2}
        {   IS >=                      } {index-name-2          }
        {   IS <=                      }
        { |---------------|            }
        { | IS UNEQUAL TO |            }
        { | EQUALS        |            }
        { | EXCEEDS       |            }
          |---------------|
```

|The format for a relation condition involving boolean expressions is as follows:|

```
|-------------------------------------------------------------------|
|                      {IS [NOT] EQUAL TO}                          |
|                      {IS [NOT] =       }                          |
| boolean-expression-1 {                 } boolean-expression-2     |
|                      {IS UNEQUAL TO    }                          |
|                      {EQUALS           }                          |
|-------------------------------------------------------------------|
```

|The format for a relation condition involving pointer expressions is as follows:|

```
|-----------------------------------------------------------|
| {ADDRESS OF identifier-3}                                 |
| {identifier-4           }                                 |
| {NULL                   }                                 |
|           {IS [NOT] EQUAL TO}                             |
|           {IS [NOT] =       } {ADDRESS OF identifier-5}   |
|           {                 } {identifier-6           }   |
|           {IS UNEQUAL TO    } {NULL                   }   |
|           {EQUALS           }                             |
|-----------------------------------------------------------|
```

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

The relational operators specify 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 given as follows:

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 TO IS [NOT] =	Equal to or not equal to
IS GREATER THAN OR EQUAL TO IS >=	Greater than or equal to
IS LESS THAN OR EQUAL TO IS <=	Less than or equal to
|EQUALS	Equal to
IS UNEQUAL TO	Not equal to
EXCEEDS	Greater than|

1. Comparison of Numeric Operands

   For operands whose class is numeric, a comparison is made with respect to the algebraic value of the operands. The length of the literal or arithmetic expression operands, in terms of the 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.

2. Comparison of Non-numeric Operands

   For non-numeric operands, or one numeric and one non-numeric operands, a comparison is made with respect to a specified collating sequence of characters. (See "OBJECT-COMPUTER Paragraph", Chapter 7). If one of the operands is specified as numeric, it must be an integer data item or an integer literal and:

   a. If the non-numeric operand is an elementary data item or a non-numeric 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 content of this alphanumeric data item were then compared to the non-numeric operand. (See the "MOVE Statement" in Chapter 12, and the "PICTURE Clause" character 'P' in Chapter 9.)

   b. If the non-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 content of this group item were then compared to the non-numeric operand. (See the "MOVE Statement" in Chapter 12, and the "PICTURE Clause" character 'P' in Chapter 9.)

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

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

   There are two cases to consider: operands of equal size and operands of unequal size.

   a. 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 corresponding characters are equal.

      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.

b. 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.

3. Comparisons Involving Index-Names and/or Index Data Items.

Relation tests may be made only between:

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

b. An index-name and a numeric data item or a numeric literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal. |If the data item or the literal are not numeric then the result will be undefined.|

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

|4. Comparison of boolean operands

A comparison of boolean operands is made regardless of their usage. If the operands are of equal size, comparison effectively proceeds by comparing boolean characters in corresponding boolean character positions starting from the high order end and continuing until either a pair of unequal boolean 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 boolean characters compare equally through the last pair, when the low order end is reached. If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by sufficient boolean character zeros to make the operands of equal size.

5. Comparison of pointer operands

Identifier-4 and identifier-6 must be described with the USAGE IS POINTER clause.|

10.5.1.2 Class Condition

The class condition determines whether the operand is numeric, alphabetic, alphabetic-lower, alphabetic-upper, |boolean|, or contains only the characters in the set of characters specified by the CLASS clause as defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION. The class of an operand is determined as follows:

- An operand is numeric if it consists entirely of the characters 0, 1, 2, 3, ... , 9, with or without an operational sign.
- An operand is alphabetic if it consists entirely of the uppercase letters A, B, ... , Z, space or the lower-case letters a, b, ... , z, space or any combination of the uppercase and lower-case letters and spaces.
- An operand is alphabetic-lower if it consists entirely of the lower-case letters a, b, c, ... z, and space.
- An operand is alphabetic-upper if it consists entirely of the uppercase letters A, B, C, .. , Z, and space.
- An operand is Boolean if it consists entirely of the characters 0 and 1.|

An operand is in conformance with class-name-1 if it consists entirely of the characters listed in the definition of class-name-1 in the SPECIAL-NAMES paragraph.

The general format for the class condition is as follows:

```
                          {  NUMERIC          }
                          {  ALPHABETIC       }
                          {  ALPHABETIC-LOWER }
identifier-1 is [NOT]     {  ALPHABETIC-UPPER }
                          { |---------------| }
                          { |  BOOLEAN      | }
                          { |---------------| }
                          { class-name-1      }
```

Identifier-1 must reference a data item whose usage is explicitly or implicitly DISPLAY. If identifier-1 is a function-identifier, it must reference an alphanumeric function.

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 non-numeric.

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 content is 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 content is 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 '-'. Valid sign(s) for data items not described with the SIGN IS SEPARATE clause (see Chapter 9) are '+' and '-' overpunched in the trailing location.

The ALPHABETIC test cannot be used with an item whose data description describes the item as numeric. The result of the test is true if the content of the data item referenced by identifier-1 consists entirely of alphabetic characters.

The ALPHABETIC-LOWER test cannot be used with an item whose data description describes the item as numeric. The result of the test is true if the content of the data item referenced by identifier-1 consists entirely of the lower-case alphabetic characters a through z and space.

The ALPHABETIC-UPPER test cannot be used with an item whose data description describes the item as numeric. The result of the test is true if the content of the data item referenced by identifier-1 consists entirely of the uppercase alphabetic characters A through Z and space.

The BOOLEAN test must not be used with an item whose data description describes the item as alphabetic or numeric. The result of the test is true if the contents of the data item referenced by identifier-1 consist entirely of boolean characters.

The class-name-1 test must not be used with an item described as numeric.

10.5.1.3 Condition-name Condition (Conditional Variable)

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with condition-name-1. The general format for the condition-name condition is as follows:

```
condition-name-1
```

If condition-name-1 is associated with a range or ranges of values, then the conditional variable is tested to determine whether or not its value falls in this range including the end values.

The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

The result of the test is 'true' if one of the values corresponding to the condition-name-1 equals the value of its associated conditional variable.

10.5.1.4 Switch-status Condition

A switch-status condition determines the 'on' or 'off' status of a JCL switch. The switch 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-1
```

The result of the test is 'true' if the switch is set to the specified position corresponding to condition-name-1.

10.5.1.5 Sign Condition

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

```
                                    { POSITIVE }
arithmetic-expression-1 IS [NOT] { NEGATIVE }
                                    { ZERO     }
```

When used, NOT and the next key word specify one sign condition that defines the algebraic test to be executed for truth value; e.g., NOT ZERO is a truth test for a non-zero (positive or negative) value. An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero.

Arithmetic-expression-1 must contain at least one reference to a variable.

10.5.2 Complex Conditions

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

The logical operators and their meanings are:

Logical Operator	Meaning
AND	Logical conjunction; the truth value is 'true' if both of the conjoined conditions are true; 'false' if one or both of the conjoined conditions is false.
OR	Logical inclusive OR; the truth value is 'true' if one or both of the included conditions is true; 'false' if both included conditions are false.
NOT	Logical negation or reversal of truth value; the truth value is 'true' if the condition is false; 'false' if the condition is true.

The logical operators must be preceded by a space and followed by a space.

10.5.2.1 Negated Conditions

A condition is negated by use of the logical operator 'NOT' which reverses the truth value of the condition to which it is applied. Thus, the truth value of a negated condition is 'true' if and only if the truth value of the condition being negated is 'false'; the truth value of a negated condition is 'false' if and only if the truth value of the condition being negated is 'true'. Including a negated condition in parentheses does not change its truth value.

The general format for a negated condition is

```
NOT condition-1
```

10.5.2.2 Combined Conditions

A combined condition results from connecting conditions with one of the logical operators 'AND' or 'OR.' The general format of a combined condition is

```
                { AND }
condition-1 { { }     } condition-2 } ...
                { OR  }
```

10.5.2.3 Precedence of Logical Operators and Use of Parentheses

In the absence of the relevant parentheses in a complex condition, the precedence (i.e., binding power) of the logical operators determines the conditions to which the specified logical operators apply and implies the equivalent parentheses. The order of precedence is 'NOT', 'AND', 'OR'. Thus, specifying 'condition-1 OR NOT condition-2 AND condition-3' implies and is equivalent to specifying 'condition-1 OR ((NOT condition-2) AND condition 3)'.

Where parentheses are used in a complex condition, they determine the binding of conditions to logical operators. Parentheses can, therefore, be used to depart from the normal precedence of logical operators as specified above. Thus, the example complex condition above can be given a different meaning by specifying it as '(condition-1 OR (NOT condition-2)) AND condition-3'. (See "Order of Evaluation of Conditions", this chapter.)

The table below indicates the ways in which conditions and logical operators may be combined and parenthesized. There must be a one-to-one correspondence between left and right parentheses such that each left parenthesis is to the left of its corresponding right parenthesis.

Table 10-3. Combinations of Conditions, Operators, Parentheses

Given the following element	In a conditional expression: May element be first?	In a conditional expression: May element be last?	In a left-to-right sequence of elements: Element, when not first, may be immediately preceded by only	In a left-to-right sequence of elements: Element, when not last, may be immediately followed by only
simple condition	Yes	Yes	OR, NOT, AND, (	OR, AND,)
OR or AND	No	No	simple-condition,)	simple condition, NOT, (
NOT	Yes	No	OR, AND, (	simple condition, (
(	Yes	No	OR, NOT, AND, (	simple-condition, NOT, (
)	No	Yes	simple-condition,)	OR, AND,)

Thus, the element pair 'OR NOT' is permissible while the pair 'NOT OR' is not permissible; the pair 'NOT (' is permissible while 'NOT NOT' is not permissible.

10.5.3 Abbreviated Combined Relation Condition

When simple or negated simple relation conditions are combined with logical connectives in a consecutive sequence such that a succeeding relation condition contains a subject or subject and relational operator that is common with the preceding relation condition, and no parentheses are used within such a consecutive sequence, any relation condition except the first may be abbreviated by the omission of the subject of the relation condition, or the omission of the subject and relational operator of the relation condition.

The format for an abbreviated combined relation condition is:

```
                     { AND }
relation-condition {{      } [NOT] [relational-operator] object}...
                     { OR  }
```

Within a sequence of relation conditions both of the above forms of abbreviation may be used. The effect of using such abbreviations is as if the last preceding stated subject were inserted in place of the omitted subject, and the last stated relational operator were inserted in place of the omitted relational operator. The result of such implied insertion must comply with the rules given in the table "Combinations of Conditions, Operators, Parentheses" above. The insertion of an omitted subject and/or relational operator terminates once a complete simple condition is encountered within a complex condition.

The interpretation applied to the case of the word NOT in an abbreviated combined relation condition is as follows:

If the word immediately following NOT is GREATER, >, LESS, <, EQUAL, =, then the NOT participates as part of the relational operator; otherwise the NOT is interpreted as a logical operator and, therefore, the implied insertion of subject or relational operator results in a negated relation condition.

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

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

10.5.4 Order of Evaluation of Conditions

Parentheses, both explicit and implicit, denote a level of inclusiveness within a complex condition. Two or more conditions connected by only the logical operator 'AND' or only the logical operator 'OR' at the same level of inclusiveness establish a hierarchical level within a complex condition. Thus, an entire complex condition may be considered to be a nested structure of hierarchical levels with the entire complex condition itself being the most inclusive hierarchical level. Within this context, the evaluation of the conditions within an entire complex condition begins at the left of the entire complex condition and proceeds according to the following rule recursively applied where necessary:

1. The constituent connected conditions within a hierarchical level are evaluated in order from left to right, and evaluation of that hierarchical level terminates as soon as a truth value for it is determined regardless of whether all the constituent connected conditions within that hierarchical level have been evaluated.

2. Values are established for arithmetic expressions and function if and when the conditions containing them are evaluated. Similarly, negated conditions are evaluated if and when it is necessary to evaluate the complex condition that they represent. (See "Formation and Evaluation Rules" above.)

10.6 CATEGORIES OF STATEMENTS

Category	Verbs
	{ ADD
	{ COMPUTE
Arithmetic	{ DIVIDE
	{ |EXAMINE| (TALLYING)
	{ INSPECT (TALLYING)
	{ MULTIPLY
	{ SUBTRACT
|Boolean|	{ |COMPUTE|
	{ COPY
Compiler directing	{ REPLACE
	{ USE
	{ ADD (SIZE ERROR)
	{ CALL (ON EXCEPTION/OVERFLOW)
	{ COMPUTE (SIZE ERROR)
	{ DELETE (INVALID KEY)
	{ DIVIDE (SIZE ERROR)
	{ EVALUATE
	{ IF
	{ MULTIPLY (SIZE ERROR)
	{ READ (AT END or INVALID KEY)
Conditional	{ RECEIVE (NO DATA)
	{ RETURN
	{ REWRITE (INVALID KEY)
	{ SEARCH
	{ START (INVALID KEY)
	{ STRING (ON OVERFLOW)
	{ SUBTRACT (SIZE ERROR)
	{ UNSTRING (ON OVERFLOW)
	{ WRITE (INVALID KEY or END-OF-PAGE)

	{ ACCEPT (DATE, DAY, DAY-OF-WEEK, or TIME)
	{ ACCEPT (MESSAGE COUNT)
	{ |EXAMINE|
	{ INITIALIZE
	{ INSPECT (CONVERTING)
	{ INSPECT (REPLACING)
Data Movement	{ MOVE
	{ SET (TO TRUE)
	{ STRING
	{ |TRANSFORM|
	{ UNSTRING
Ending	{ STOP
	{ ACCEPT (identifier)
	{ CLOSE
	{ DELETE
	{ DISABLE
	{ DISPLAY
	{ ENABLE
	{ OPEN
	{ PURGE
	{ READ
Input-Output	{ RECEIVE
	{ REWRITE
	{ SEND
	{ SET (TO ON, TO OFF)
	{ START
	{ STOP (literal)
	{ WRITE
Inter-Program Communicating	{ CALL
	{ CANCEL
No Operation	{ CONTINUE
	{ EXIT
	{ MERGE
	{ RELEASE
Ordering	{ RETURN
	{ SORT
	{ ALTER
	{ CALL
Procedure Branching	{ EXIT (PROGRAM)
	{ GO TO
	{ PERFORM

Report Writing	{ GENERATE { INITIATE { SUPPRESS { TERMINATE
Scope Delimiting	{ ADD (END-ADD) { CALL (END-CALL) { COMPUTE (END-COMPUTE) { DELETE (END-DELETE) { DIVIDE (END-DIVIDE) { EVALUATE (END-EVALUATE) { IF (END-IF) { MULTIPLY (END-MULTIPLY) { PERFORM (END-PERFORM) { READ (END-READ) { RECEIVE (END-RECEIVE) { RETURN (END-RETURN) { REWRITE (END-REWRITE) { SEARCH (END-SEARCH) { START (END-START) { STRING (END-STRING) { SUBTRACT (END-SUBTRACT) { UNSTRING (END-UNSTRING) { WRITE (END-WRITE)
Table Handling	{ SEARCH { SET (TO, UP BY or DOWN BY)

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

10.6.1 Specific Statement Formats

The specific statement formats, together with a detailed discussion of the restrictions and limitations associated with each, appear hereafter in alphabetic order.

10.7 COMMON OPTIONS AND RULES FOR STATEMENT FORMATS

The subordinate paragraphs provide a description of the common options and conditions that pertain to or appear in several different statements.

10.7.1 Intermediate Data Item

An intermediate data item is a signed numeric data item provided by the compiler that contains the results developed in the course of an arithmetic operation prior to the final result being moved to the resultant identifier, if any. This data item is 18 |(up to 30 if the compilation is run with the LEVEL = NSTD parameter) digits in length and contains the 18 (or 30 if the compilation is run with the LEVEL = NSTD parameter)| most significant digits of the result being developed during the execution of an arithmetic operation. During execution of an arithmetic operation, the magnitude of the mathematical result is maintained and all low-order digits which are truncated are considered to be zero for the remainder of this arithmetic operation. The TEMP clause of |the Default Section of the Control Division may change the above rules.|

10.7.2 The ROUNDED Phrase

If, after decimal point alignment, the number of places in the fractions 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 in the low-order position whenever the most significant digit of 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 clause for that resultant identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

If the result is described with the USAGE IS BINARY clause, the value of the result will be exactly the same as it was described with the USAGE IS DISPLAY clause.

10.7.3 The SIZE ERROR Phrase

The size error condition occurs under the following circumstances:

1. Violation of the rules for evaluation of exponentiation always terminates the arithmetic operation and always causes a size error condition. (See "Formation and Evaluation Rules" above.)

2. Division by zero always terminates the arithmetic operation and always causes a size error condition.

3. If, after radix 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. |In this case, the size error condition applies only to the final results of an arithmetic operation and does not apply to intermediate results. The algebraic value of the final result of the arithmetic operation(s) is accurate to the precision specified by the resultant identifier, except when the TEMP clause of the Default Section of the Control Division permits some loss of precision.|

   If the ROUNDED phrase is specified, rounding takes place before checking for size error.

If the ON SIZE ERROR phrase is specified and a size error condition exists after the execution of the arithmetic operations specified by an arithmetic statement, the values of the affected resultant identifiers remain unchanged from the values they had before execution of the arithmetic statement. The values of resultant identifiers for which no size error condition exists are the same as they would have been if the size error condition had not resulted for any of the resultant identifiers. After completion of the arithmetic operations, control is transferred to the imperative-statement specified in the ON SIZE ERROR phrase and execution continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the ON SIZE ERROR phrase, control is transferred to the end of the arithmetic statement and the NOT ON SIZE ERROR phrase, if specified, is ignored.

If the ON SIZE ERROR phrase is not specified and a size error condition exists after the execution of the arithmetic operations specified by an arithmetic statement, the values of the affected resultant identifiers are undefined. The values of resultant identifiers for which no size error condition exists are the same as they would have been if the size error condition had not resulted for any of the resultant identifiers. After completion of the arithmetic operations, control is transferred to the end of the arithmetic statement and the NOT ON SIZE ERROR phrase, if specified, is ignored.

If the size error condition does not exist after the execution of the arithmetic operations specified by an arithmetic statement, the ON SIZE ERROR phrase, if specified, is ignored and control is transferred to the end of the arithmetic statement or to the imperative-statement specified in the NOT ON SIZE ERROR phrase if it is specified. In the latter case, execution continues according to the rules for each statement specified in that imperative statement. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the NOT ON SIZE ERROR phrase, control is transferred to the end of the arithmetic statement.

For the ADD statement with the CORRESPONDING phrase and the SUBTRACT statement with the CORRESPONDING phrase, if any of the individual operations produces a size error condition, imperative-statement-1 in the SIZE ERROR phrase is not executed until all the individual additions or subtractions are completed.

If the result is described with the usage is binary clause and the ON SIZE ERROR phrase is specified, the size error condition and the value of the result will be exactly the same as if the result was described with the USAGE IS DISPLAY clause.

10.7.4 The CORRESPONDING Phrase

For the purpose of this discussion, identifier-1 and identifier-2 are the identifiers specified in a statement which contains the CORRESPONDING phrase.

1. Rules for valid identifiers are:

    a. All identifiers must refer to group items.

    b. The REDEFINES or OCCURS clause may be specified in the data description entry of any of the identifiers.

    c. Identifiers may be subordinate to a data description entry containing a REDEFINES or OCCURS clause.

    d. No identifier may be defined with level-number 66, with level number 77, level number 88 or with the USAGE IS INDEX clause.

    e. No identifier may be reference modified.

2. Data items subordinate to identifier-1 correspond with data items subordinate to identifier-2, if the following rules apply:

    a. Both data items must have the same data-name.

    b. All possible qualifiers for the sending data item, up to but not including identifier-1, must be identical to all possible qualifiers for the receiving data item up to but not including identifier-2.

    c. In an ADD or SUBTRACT statement, only elementary numeric data items will be considered.

    d. In a MOVE statement, the corresponding sending and/or receiving data items must be elementary. The class of any corresponding pair of data items may differ.

e. A data item with a level-number of 66 or 88 or with a data description entry containing a REDEFINES, OCCURS, USAGE IS POINTER or |USAGE IS INDEX clause| is not considered. Any data item subordinate to a data item which is not eligible for correspondence will also be ignored.

f. FILLER data items and data items subordinate thereto are ignored.

10.7.5 The Arithmetic Statements

The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements. They have several common features.

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

2. The maximum size of each operand is eighteen decimal digits, |(or thirty if the compilation is run with the LEVEL = NSTD parameter).|

3. For ADD, DIVIDE, MULTIPLY, and SUBTRACT statements, the composite of operands, which is a hypothetical data item resulting from the super-imposition of specific operands in a statement aligned on their decimal points, must not contain more than 18 decimal digits (or 30 if LEVEL 64).

4. Each arithmetic operation is evaluated using an intermediate data item provided for the result of the arithmetic operation. If the size of the result being developed is larger than this intermediate data item, truncation occurs. The content of the intermediate data item is moved to the resultant identifier according to the rules for the MOVE statement. Rounding is performed and the size error condition is determined only during this move. (See the "ROUNDED Phrase", and the "SIZE ERROR Phrase" in this chapter, and the "MOVE Statement" in Chapter 12.)

10.7.6 Overlapping Operands

When a sending and a receiving item in any statement share a part or all of their storage areas, yet are not defined by the same data description entry, the result of the execution of such as statement is undefined. In addition, the results are undefined for some statements in which sending and receiving items are defined by the same data description entry. These cases are addressed in the general rules associated with those statements.

10.7.7 Multiple Results in Arithmetic Statements

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

1. A statement whose execution accesses all data items that are part of the initial evaluation of the statement, performs any necessary arithmetic or combining of these data items and stores the result of this operation in a temporary location. See the individual statements for the rules indicating which items are part of the initial evaluation.

2. A sequence of statements whose execution transfers or combines the value in this temporary location with each single resulting data item. These statements are considered to be written in the same left-to-right sequence that the multiple results are specified.

The result of the statement

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

is equivalent to

```
ADD a, b, c GIVING temp
ADD temp TO c
ADD temp TO d(c)
ADD temp TO e
```

and the result of the statement

```
MULTIPLY a (i) BY i, a (i)
```

is equivalent to

```
MOVE a (i) TO temp
MULTIPLY temp BY i
MULTIPLY temp BY a (i)
```

in both cases, 'temp' is an intermediate result item provided by the compiler.

10.7.8 Incompatible Data

Except for the class condition, when the content of a data item is referenced in the Procedure Division and the content of that data item is not compatible with the class specified for that data item by its PICTURE clause or function definition, then the result of such a reference is undefined. (See "Class Condition", this chapter.)

10.7.9 The INVALID KEY Condition

The format of the INVALID KEY phrase is:

```
[INVALID KEY imperative-statement-1]
[NOT INVALID KEY imperative-statement-2]
```

The invalid key condition can occur as a result of the execution of a DELETE, READ, REWRITE, START or WRITE statement. When the invalid key condition occurs, execution of the input-output statement which recognized the condition is unsuccessful and the file is not affected. (See the "DELETE Statement" in Chapter 11, the "READ Statement" and the "REWRITE Statement" in Chapter 12, the "START Statement" and the WRITE Statement in Chapter 13.)

If the invalid key condition exists after the execution of the input-output operation specified in an input-output statement, the following actions occur in the order shown:

1. The I-O status of the file connector associated with the statement is set to a value indicating the invalid key condition. (See "I-O Status" of the statement concerned.)

2. If the INVALID KEY phrase is specified in the input-output statement, any USE AFTER EXCEPTION procedure associated with the file connector is not executed and control is transferred to the imperative statement specified in the INVALID KEY phrase. Execution then continues according to the rules for each statement specified in that imperative statement. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative statement specified in the INVALID KEY phrase, control is transferred to the end of the input-output statement and the NOT INVALID KEY phrase, if specified, is ignored.

3. If the INVALID KEY phrase is not specified in the input-output statement, a USE AFTER EXCEPTION procedure must be associated with the file connector and that procedure is executed and control is transferred according to the rules of the USE statement. The NOT INVALID KEY phrase, if specified, is ignored.

If the invalid key condition does not exist after the execution of the input-output operation specified by an input-output statement, the INVALID KEY phrase, if specified, is ignored. The I-O status of the file connector associated with the statement is updated and the following actions occur in the order shown:

1. If an exception condition which is not an invalid key condition exists, control is transferred according to the rules of the USE statement following the execution of any USE AFTER EXCEPTION procedures associated with the file connector. (See "I-O Status" of the statement concerned.)

2. If no exception condition exists, control is transferred to the end of the input-output statement or to the imperative statement specified in the NOT INVALID KEY phrase if it is specified. In the latter case, execution continues according to the rules for each statement specified in that imperative statement. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative statement specified in the NOT INVALID KEY phrase, control is transferred to the end of the input-output statement.

10.7.10 The AT END Condition

The format of the AT END phrase is:

```
[AT END imperative-statement-1]

[NOT AT END imperative-statement-2]
```

The at end condition can occur as a result of the execution of a READ, or RETURN statement.

If an at end condition does not occur at the time of execution of a statement containing the NOT AT END phrase, control is transferred to imperative-statement-2 in the NOT AT END phrase after updating of the I-O status associated with the file-name. If an exception condition which is not an at end condition occurs, this transfer of control takes place after execution of the procedures, if any, specified by the USE AFTER STANDARD EXCEPTION statement applicable to the file-name.

10.7.11 The FROM Option

The format of the FROM phrase is:

```
record-name-1 FROM identifier-1
```

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

The result of the execution of a RELEASE, REWRITE or WRITE statement with the FROM phrase is equivalent to the execution of the following statements in the order specified:

1. The statement:

   ```
   MOVE identifier-1 TO record-name-1
   ```

   according to the rules specified for the MOVE statement.

2. The same RELEASE, REWRITE or WRITE statement without the FROM phrase.

After the execution of the RELEASE, REWRITE or 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-1 is not available except as specified by the SAME RECORD AREA clause.

10.7.12 The INTO Option

The format of the INTO phrase is:

```
file-name-1 INTO identifier-1
```

The storage area associated with identifier-1 and the record area associated with file-name-1 must not be the same storage area.

The INTO phrase may be specified in READ or RETURN statements:

1. If only one record description is subordinate to the file description entry, or

2. If all record-names associated with file-name-1 and the data item associated with identifier-1 describe a group item or an elementary alphanumeric item.

The results of the execution of a READ or RETURN statement with the INTO phrase are equivalent to the application of the following rules in the order specified:

1. The execution of the same READ or RETURN statement without the INTO phrase.

2. The current record is moved from the record area to the area specified by identifier-1 |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.| The size of the current record is determined by rules specified for the RECORD clause. If the file description entry contains a RECORD VARYING clause, the implied move is a group move.

   The implied MOVE statement does not occur if the execution of the READ or RETURN statement was unsuccessful. Any subscripting associated with identifier-1 is evaluated after the record has been read and immediately before it is moved to the data item. The record is available in both the record area and the data referenced by identifier-1.

11. Procedure Division - Statements (ACCEPT to GO TO)

This chapter describes the statements from ACCEPT to GO TO (inclusive). The statements concerned are as follows:

- ACCEPT
- ADD
- ALTER
- |ASSIGN|
- CALL
- CANCEL
- CLOSE
- COMPUTE
- CONTINUE
- DELETE
- DISABLE
- DISPLAY
- DIVIDE
- ENABLE
- EVALUATE
- |EXAMINE|
- EXIT
- GENERATE
- GO TO

11.1 ACCEPT

Description

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

Format 4 allows you to obtain the file-literal associated to a file.

Format 1

```
                            {    mnemonic-name-1      }
                            { |---------------------| }
                            { |  SYSIN               | }
ACCEPT identifier-1 [FROM   { | [ALTERNATE] CONSOLE  | }
                            { |  ALTERNATE-CONSOLE   | }
                            { |  TERMINAL            | }
                              |---------------------|
```

Format 2

```
                           {DATE       }
                           {DAY        }
ACCEPT identifier-2 FROM   {           }
                           {DAY-OF-WEEK}
                           {TIME       }
```

Format 3

```
ACCEPT cd-name-1 MESSAGE COUNT
```

Format 4

```
|----------------------------------------|
| ACCEPT identifier-3 FROM FILE file-name |
|----------------------------------------|
```

Syntax Rules

1. Mnemonic-name-1 in format 1 must also be specified in the SPECIAL-NAMES paragraph of the Environment Division, and must be associated with CONSOLE, ALTERNATE-CONSOLE, |ALTERNATE CONSOLE,| TERMINAL, or SYSIN.

2. Cd-name-1 must reference an input CD.

|3. Identifier-2 must not reference a boolean data item.

4. Identifier-3 must be described as an alphanumeric item.|

General Rules

Format 1

1. The ACCEPT statement causes the transfer of data from the hardware device. The data replaces the content of the data item referenced by identifier-1. Conversion is applied to the transfer, |depending on the suffixed device name. (See "Legible Equivalent" in Chapter 3, and "Default Section" in Chapter 5).|

2. If mnemonic-name-1 is associated with SYSIN, or if SYSIN is explicitly specified, data is accepted from the file whose internal-file-name is H_RD.

   If mnemonic-name-1 is associated with CONSOLE, |or if CONSOLE is explicitly specified,| data is accepted from the main operator console.

   If mnemonic-name-1 is associated with ALTERNATE-CONSOLE |or ALTERNATE CONSOLE, or if ALTERNATE-CONSOLE or ALTERNATE CONSOLE are explicitly specified,| data is accepted from the alternate operator console specified in the CONSOLE JCL statement. If no alternate console is specified, data is accepted from the console from which the job is submitted.

   If mnemonic-name-1 is associated with TERMINAL, |or if TERMINAL is explicitly specified,| data is accepted from the alternate operator console specified in the CONSOLE JCL statement. If no alternate console is specified, data is accepted from the console from which the job is submitted. If the program is interactively run from an I0F terminal, data is accepted from that very terminal.

   To determine the size of a data transfer, a required size is defined. The required size depends on the category of the receiving data item and the suffix implicitly or explicitly appended to the hardware name associated with mnemonic-name-1 |or specified in the FROM phrase|.

   a. The suffix is -0: The required size is the number of character positions in the receiving area.

   b. The suffix is -2 and the receiving data item is numeric: The number of characters received is considered to be the required size; when the hardware name associated with mnemonic-name-1 |or specified in the FROM phrase| is SYSIN and if the standard method applies (see the *COBOL 85 User's Guide*), as many records as necessary to get a non blank character are read, if end-of-file is reached, the program is in error.

   c. Other cases: The required size is the number of characters in the legible input equivalent.

3. If the size of the transferred data is equal to the required size, it is stored in the receiving data item.

4. If the size of the transferred data is not equal to the required size, then:

   a. If the required size exceeds the total size of transferred data, then additional data is requested if the hardware name associated with mnemonic-name-1 |[or specified in the FROM phrase]| is SYSIN and the standard method applies (see the *COBOL 85 User's Guide*). After additional data is transferred, rules 3 and 4 apply. If end-of-file is reached or if the hardware name is not SYSIN or if the hardware name is SYSIN and the console method applies (see the *COBOL 85 User's Guide*), the transferred data is padded with as many characters as necessary to get the required size. The padding characters are zeros |if the receiving data item is of category boolean or if it is usage POINTER or| if the suffix implicitly or explicitly appended to the hardware name is -X, or blank characters otherwise.

   b. If the size of the transferred data exceeds the required size, only the leftmost characters of the transferred data are used to be stored in the receiving data item. If the remaining characters of the transferred data are all blank characters or if the suffix implicitly or explicitly appended to the hardware name associated with mnemonic-name-1 |[or specified in the FROM phrase]| is -0, the characters in excess are ignored, else the program is in error.

5. If the FROM option is not given, data is accepted from the file whose internal-file-name is H_RD, i.e. the standard device is that implied by SYSIN. |However, another default may be specified in the ACCEPT clause of the Default Section in the Control Division.|

Format 2

6. The ACCEPT statement causes the information requested to be transferred to the data item specified by identifier-2 according to the rules for the MOVE statement. (See the "MOVE Statement" in Chapter 12.) DATE, DAY, DAY-OF-WEEK and TIME are conceptual data items and, therefore, are not described in the COBOL program.

7. DATE is composed of the data elements year of century, month of year, and day of month. The sequence of the data element codes is from high order to low order (left to right), year of century, month of year, and day of month. Therefore, February 14, 1982 would be expressed as 820214. DATE, when accessed by a COBOL program, behaves as if it had been described in a COBOL program as an unsigned elementary numeric integer data item six digits in length.

8. DAY is composed of the data elements year of century and day of year. The sequence of the data element codes is from high order to low order (left to right) year of century, day of year. Therefore, February 14, 1982 would be expressed as 82045. DAY, when accessed by a COBOL program, behaves as if it had been described in a COBOL program as an unsigned elementary numeric integer data item five digits in length.

9. TIME is composed of the data elements hours, minutes, seconds and hundredths of a second. TIME is based on elapsed time after midnight on a 24-hour clock basis - thus, 2:41 p.m. would be expressed as 14410000. TIME, when accessed by a COBOL program, behaves as if it had been described in a COBOL program as an unsigned elementary numeric integer data item eight digits in length. The minimum value of TIME is 00000000; the maximum value is 23595999.

10. DAY-OF-WEEK is composed of a single data element whose content represents the day of the week. DAY-OF-WEEK, when accessed by a COBOL program, behaves as if it had been described in the COBOL program as an unsigned elementary numeric integer data item one digit in length. In DAY-OF-WEEK, the value 1 represents Monday, 2 represents Tuesday, ..., 7 represents Sunday.

Format 3

11. The ACCEPT statement causes the MESSAGE COUNT field specified for cd-name-1 to be updated to indicate the number of complete messages that exist in the queue structure designated by the contents of the data items specified by data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name-1.

12. Upon execution of the ACCEPT MESSAGE COUNT statement, the content of the area specified by a Communication Description entry must contain at least the name of the symbolic queue to be tested. Testing the condition causes the content of the data items referenced by data-name-10 (STATUS KEY) and data-name-11 (MESSAGE COUNT) of the area associated with the Communication Description entry to be appropriately updated.

Format 4

|13. A file-literal corresponding to the current assignment of file-name is built and then moved to identifier-3. If file-name is not assigned, identifier-3 is set to spaces.

14. Information such as access, end, abend, ... if any, are lost.|

15. If file-name is defined with an ORGANIZATION clause that specifies QUEUED, the member name that could have been determined by any execution of an ASSIGN verb containing the TO MEMBER phrase is not part of the value returned by a format 4 ACCEPT statement. This information is got when identifier-1 is used in a format 1 ASSIGN statement.|

11.2 ADD

Description

The ADD statement sums two or more numeric operands to be and stores the result.

Format 1

```
    {identifier-1}
ADD {            }... TO {identifier-2 [ROUNDED]}...
    {literal-1   }

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-ADD]
```

Format 2

```
    {identifier-1}       {identifier-2}
ADD {            }... TO {            }
    {literal-1   }       {literal-2   }

            GIVING {identifier-3 [ROUNDED]}...

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-ADD]
```

Format 3

```
    {CORRESPONDING}
ADD {              } identifier-1 TO identifier-2 [ROUNDED]
    {CORR          }

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-ADD]
```

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. In Format 3, each identifier must refer to a group item.

2. Each literal must be a numeric literal.

3. The composite of operands must not contain more than 18 digits |(up to 30 if the compiler is run with the LEVEL=NSTD parameter).| In format 1 the composite of operands is determined by using all of the operands in a given statement. 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 . In format 3 the composite of operands is determined separately for each pair of corresponding data items.

4. CORR is an abbreviation for CORRESPONDING.

General Rules

1. If format 1 is used, the values of the operands preceding the word TO are added together and the sum is stored in a temporary data item. The value in this temporary data item is added to the value of the data item referenced by identifier-2, storing the result into the data item referenced by identifier-2, and repeating this process for each occurrence of identifier-2 in the left-to-right order in which identifier-2 is specified.

2. If format 2 is used, the values of the operands preceding the word GIVING are added together, then the sum is stored as the new content of each data item referenced by identifier-3.

3. If format 3 is used, data items in identifier-1 are added to and stored in corresponding data items in identifier-2.

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

5. Additional rules and explanations relative to this statement are given under the appropriate paragraphs. (See "Scope of Statements", "Intermediate Data Item", the "ROUNDED Phrase", the "Arithmetic Statements", "Overlapping Operands", the "SIZE ERROR Phrase", the "CORRESPONDING Phrase", and "Multiple Results in Arithmetic Statements" in Chapter 10.)

11.3 ALTER

Description

The ALTER statement modifies a predetermined sequence of operations.

The ALTER statement is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

Format

```
ALTER {procedure-name-1 TO [PROCEED TO] procedure-name-2}...
```

Syntax Rules

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

General Rules

1. Execution of the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, so that subsequent executions of the modified GO TO statement 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.

2. A GO TO statement in a section with a segment-number greater than 49 must not be referred to by an ALTER statement in a section with a different segment-number.

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

11.4 ASSIGN

Description

|The ASSIGN statement associates a member name to a queued file, or assigns a file.|

Format 1

```
|-----------------------------------------------------|
|ASSIGN file-name-1                                   |
|                                                     |
|           {[NOT] GREATER THAN       }               |
|           {[NOT] LESS THAN          }               |
|           {[NOT] EQUAL TO           }               |
|           { GREATER THAN OR EQUAL TO} {identifier-1}|
|TO MEMBER { LESS THAN OR EQUAL TO   } {literal-1   }|
|           {[NOT] >                  } {ACTUAL     }|
|           {[NOT] <                  }               |
|           {[NOT] =                  }               |
|           { >=                      }               |
|           { <=                      }               |
|-----------------------------------------------------|
```

Format 2

```
|-----------------------------------------|
|                           {identifier-2} |
|ASSIGN file-name-2 TO FILE {literal-2   } |
|                           {file-name-3 } |
|-----------------------------------------|
```

Syntax Rules

|1. File-name-1 must be described with an ORGANIZATION clause that specifies QUEUED, with no ifn-suffix except possibly -MSD.

2. Identifier-1 must be described as an alphanumeric item.

3. Literal-1 must be a non-numeric literal.

4. The description of file-name-1 or file-name-2 must specify a file literal in the ASSIGN phrase of the SELECT entry.

5. Identifier-2 must be described as an alphanumeric item.

6. Literal-2 must be a non numeric literal.|

General Rules

Format 1

|1. File-name-1 must not be in the open state.

2. File-name-1 must have been assigned to a queued file, either by JCL or by a previously executed Format 2 ASSIGN statement.

3. The file connector associated to file-name-1 contains the name of the member being assigned to file-name-1, let's call this name the "actual member name". The initial value of the actual member name or its value after the execution of a format 2 ASSIGN file-name-1 statement is ALL LOW-VALUES. The execution of the format 1 ASSIGN statement records in actual member name the name of the member of the queued file assigned to file-name-1 which meets the condition specified with regard to the compared name. The compared name is the value of identifier-1 if identifier-1 is specified, or literal-1 if literal-1 is specified, or the current value of actual member name if ACTUAL is specified. Comparison is done according to the NATIVE collating sequence.

   If TO MEMBER GREATER THAN or TO MEMBER > is specified, actual member name is set to the least member name greater than the compared name.

   If TO MEMBER LESS THAN or TO MEMBER < is specified, actual member name is set to the greatest member name less than the compared name.

   If TO MEMBER EQUAL TO or TO MEMBER = is specified, actual member name is set to the member name equal to the compared name.

   If TO MEMBER NOT GREATER THAN or TO MEMBER NOT > or TO MEMBER LESS THAN OR EQUAL TO or TO MEMBER <= is specified, actual member name is set to the greatest member name less than or equal to the compared name.

   If TO MEMBER NOT LESS THAN or TO MEMBER NOT < or TO MEMBER GREATER THAN OR EQUAL TO or TO MEMBER >= is specified, actual member name is set to the least member name greater than or equal to the compared name.

   If TO MEMBER NOT EQUAL TO or TO MEMBER NOT = is specified, actual member name is set to any member name different from the compared name.

   If no member name meets the required condition, actual member name is set to ALL LOW-VALUES.

4. When an OPEN statement is executed for a file, and the file is a queued file, it is the member whose name is the actual-member-name recorded in the file connector associated with the file which is opened. If such a member does not exist, when a OPEN statement without the OUTPUT phrase associated with this file is executed, the OPEN is unsuccessful, and the file status data item, if any, is set to "9W".|

Format 2

|5. File-name-2 and file-name-3 must not be in the open state.

6. The value of literal-2 or identifier-2 must be of the form of a file literal possibly followed by assignment parameters. The syntax of the contents of literal-2 or identifier-2 is the same as that of the literal in the ASSIGN clause of the File-Control entry except that no internal-file-name must be specified. (See "FILE-CONTROL ENTRY", Chapter 7).

7. Previously to any other processing or checking, file-name-2 is deassigned if previously assigned.

8. If identifier-2 or literal-2 contains no valid file-literal, the ASSIGN statement is unsuccessful.|

9. If file-name-3 is specified, then file-name-2 is assigned the same as file-name-3 is. If file-name-3 is declared as QUEUED, any member name which it may have been assigned to using the format 1 ASSIGN verb is not taken into account, in that case file-name-2 should also be declared as QUEUED.|

10. The effect of an ASSIGN statement is to record assignment information rather than actually assign the file. This will take place when a subsequent OPEN or Format 1 ASSIGN statement is executed just before the opening operation begins.|

11. When an OPEN statement is executed for a file, and the last executed ASSIGN statement for that file was unsuccessful, the OPEN statement is unsuccessful, and the file status data item, if any, is set to "9M".|

11.5 CALL

Description

The CALL statement causes control to be transferred from one object program to another, within the run unit.

Format

```
     {identifier-1}
CALL {            } [USING
     {literal-1   }
                      |------------|
   {[BY REFERENCE] {[ | ADDRESS OF | ] identifier-2}...}
   {                  |------------|                   }
   {                                                   }
   {            {   identifier-3             }         }... ]
   {            { |--------------------------| }       }
   { BY CONTENT { | literal-2                  | }...    }
                { | arithmetic-expression-1 | }
                { | boolean-expression-1    | }
                  |--------------------------|

   |----------------------|
   | [GIVING identifier-4 |
   |----------------------|

       {EXCEPTION}
    [ON {         } imperative-statement-1]
       {OVERFLOW }

            {   EXCEPTION  }
    [NOT ON { |----------| } imperative-statement-2]
            { | OVERFLOW | }
              |----------|

    [END-CALL]
```

Syntax Rules

1. Literal-1 must be a non-numeric literal such that it can be a program-name.

2. Identifier-1 must be defined as an alphanumeric data item such that its value can be a program-name.

3. Each of the operands in the USING phrase must have been defined as a data item in the File Section, Working-Storage Section, |Constant Section,| Communication Section, or Linkage Section, and must be a level 01 or 77 data item or an elementary data item, |or a group data item. If identifier-2 references a data item whose usage is bit, its level-number must be 01 or 77.|

4. The same data-name may be referenced more than once in a CALL statement.

5. Identifier-2 must not be a function-identifier.

|6. The words EXCEPTION and OVERFLOW are synonymous and may be used interchangeably.

7. Literal-2, arithmetic-expression-1 or boolean-expression-1 may be used only when the called program is not externally compiled to the calling program.|

|8. Identifier-4 must reference either

   a. an elementary item of USAGE COMP-1, COMP-2, COMP-9, COMP-10 or POINTER,

   b. an elementary item of USAGE ALPHABETIC or ALPHANUMERIC of one character in length.|

General Rules

1. Literal-1 or the content of the data item referenced by identifier-1 is the name of the called program. The program in which the CALL statement appears is the calling program. If the program being called is a COBOL program, literal-1 or the content of the data item referenced by identifier-1 must contain the program-name contained in the PROGRAM-ID paragraph of the called program.

2. If, when a CALL statement is executed, the program specified by the CALL statement is made available for execution, control is transferred to the called program. After control is returned from the called program, the ON EXCEPTION phrase, if specified, is ignored and control is transferred to the end of the CALL statement or, if the NOT ON EXCEPTION phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the CALL statement.

3. If it is determined, when a CALL statement is executed, that the program specified by the CALL statement cannot be made available for execution at the time, one of the two actions listed below will occur.

   a. If the ON EXCEPTION phrase is specified in the CALL statement, control is transferred to imperative-statement-1. Execution then continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the CALL statement and the NOT ON EXCEPTION phrase, if specified, is ignored.

   b. If the ON EXCEPTION phrase is not specified in the CALL statement, the NOT ON EXCEPTION phrase, if specified, is ignored and the run unit aborts.

4. Two or more programs in the run unit may have the same program-name, and the reference in a CALL statement to such a program-name is resolved by using the scope of names conventions for program-names. (See "Conventions for Program-Names" in Chapter 3.)

   For example, when only two programs in the run unit have the same name as that specified in a CALL statement:

   a. One of those two programs must also be contained directly or indirectly either within the separately compiled program which includes that CALL statement or within the separately compiled program which itself directly or indirectly contains the program which includes that CALL statement, and

   b. The other of those two programs must be a different separately compiled program.

   The mechanism used in this example is as follows:

   a. If one of the two programs having the same name as that specified in the CALL statement is directly contained within the program which includes that CALL statement, that program is called.

   b. If one of the two programs having the same name as that specified in the CALL statement possesses the common attribute and is directly contained within another program which directly or indirectly contains the program which includes the CALL statement, that common program is called unless the calling program is contained within that common program.

   c. Otherwise, the separately compiled program is called.

5. If the called program does not possess the initial attribute, it, and each program directly or indirectly contained within it, is in its initial state 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 and each program directly or indirectly contained within it remains unchanged from its state when last exited. This includes the internal data.

6. If the called program possesses the initial attribute, it and each program directly or indirectly contained within it, is placed into its initial state every time the called program is called within a run unit.

7. Files associated with a called program's internal file connectors are not in the open mode when the program is in an initial state. (See "Initial State of a Program" in the Glossary.)

   On all other entries into the called program, the states and positioning of all such files are the same as when the called program was last exited.

8. The process of calling a program or exiting from a called program does not alter the status or positioning of a file associated with any external file connector.

9. The USING phrase may be included in the CALL statement only if there is a USING phrase in Procedure Division header of the called program and the number of operands in each USING phrase must be identical.

10. The sequence of appearance of parameters, in the USING phrase of the CALL statement and in the corresponding USING phrase in the called program's Procedure Division header determines the correspondence between the parameters used by the calling and called programs. This correspondence is positional and not by name equivalence; the first parameter in one USING phrase corresponds to the first parameter in the other, the second to the second, etc. The data description of each parameter in the CALL statement must be the same as the data description of the corresponding parameter in the USING phrase of the Procedure Division header. (See the "Procedure Division Header" in Chapter 10.)

11. The values of the parameters referenced in the USING phrase of the CALL statement are made available to the called program at the time the CALL statement is executed.

12. Both the BY CONTENT and BY REFERENCE phrases are transitive across the parameters which follow them until another BY CONTENT or BY REFERENCE phrase is encountered. If neither the BY CONTENT nor the BY REFERENCE phrase is specified prior to the first parameter, the BY REFERENCE phrase is assumed.

13. If the BY REFERENCE phrase is either specified or implied for a parameter, the object program operates as if the corresponding data item in the called program occupies the same storage area as the data item in the calling program. The description of the data item in the called program must describe the same number of character positions as described by the description of the corresponding data item in the calling program.

14. If the BY CONTENT phrase is specified or implied for a parameter, the called program cannot change the value of this parameter as referenced in the CALL statement's USING phrase, though the called program may change the value of the data item referenced by the corresponding data-name in the called program's Procedure Division header. |Previous to actually call the called program, each BY CONTENT parameter is moved to a dummy data name with the following rules:|

    |a. If the called program is in the same separately compiled program as the calling program the dummy data name has the same description as the corresponding data name in the USING phrase of the called program. The move is done according to the rules of the MOVE statement for identifier-3, or literal.2, and according to the rules for the COMPUTE statement without the ROUNDED phrase for arithmetic-expression-1 or boolean-expression-1.

    b. If the called program is in another separately compiled program than the calling program, the dummy data name is assumed to be of the same description as identifier-3 and the move is a group move.|

15. Called programs may contain CALL statements. However, a called program must not execute a CALL statement that directly or indirectly calls the calling program. If a CALL statement is executed within the range of a declarative, that CALL statement cannot directly or indirectly reference any called program to which control has been transferred and which has not completed execution.

16. The END-CALL phrase delimits the scope of the CALL statement. (See "Scope of Statements" in Chapter 10.)

|17. When the ADDRESS OF option of the USING phrase is used, the data which is available to the called program is the address of the data item referenced by identifier-2, and not the data item itself. This address is handled as if it were a data item described with the USAGE IS POINTER clause.|

|18. The GIVING phrase is used when the called program is written in C Language and return a value (External Function). After execution of the called program, the return value is made available in the data item referenced by identifier-4.|

11.6 CANCEL

Description

The CANCEL statement ensures that the next time the referenced program is called it will be in its initial state.

Format

```
        {literal-1   }
CANCEL  {            }...
        {identifier-1}
```

Syntax Rules

1. Literal-1 must be a non-numeric literal such that it can be program-name.

2. Identifier-1 must reference an alphanumeric data item such that its value can be a program-name.

General Rules

1. Literal-1 or the content of the data item referenced by identifier-1 identifies the program to be cancelled.

2. Subsequent to the execution of an explicit or implicit CANCEL statement, the program referred to therein ceases to have any logical relationship to the run unit in which the CANCEL statement appears. If the program referenced by a successfully executed explicit or implicit CANCEL statement in a run unit is subsequently called in that run unit, that program is in its initial state.

3. A program named in a CANCEL statement in another program must be callable by that other program. (See "Scope of Names" in Chapter 3, and the "CALL Statement" in Chapter 11.)

4. When an explicit or implicit CANCEL statement is executed, all programs contained within the program referenced by the CANCEL statement are also cancelled. The result is the same as if valid CANCEL statement were executed for each contained program in the reverse order in which the programs appear in the separately compiled program.

5. A program named in the CANCEL statement must not refer directly or indirectly to any program that has been called and has not yet executed an EXIT PROGRAM statement.

6. A logical relationship to a cancelled program is established only by execution of a subsequent CALL statement naming that program.

7. A called program is cancelled either by being referred to as the operand of a CANCEL statement, by the termination of the run unit of which the program is a member, or by execution of an EXIT PROGRAM statement in a called program that possesses the initial attribute.

8. No action is taken when an explicit or implicit 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 is transferred to the next executable statement following the explicit CANCEL statement.

9. The contents of data items in external data records described by a program are not changed when that program is cancelled.

10. During execution of an explicit or implicit CANCEL statement, an implicit CLOSE statement without any optional phrases is executed for each file in the open mode that is associated with an internal file connector in the program named in the explicit CANCEL statement. Any USE procedures associated with any of these files are not executed.

11.7 CLOSE

Description

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

Format

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

Syntax Rules

1. The files referenced in the CLOSE statement need not all have the same organization or access.

2. The REEL or UNIT phrase must only be used for sequential files.

General Rules

Except where otherwise stated in the general rules below, the terms 'reel' and 'unit' are synonymous and completely interchangeable in the CLOSE statement. Treatment of sequential mass storage files is logically equivalent to the treatment of a file on tape or analogous sequential media. Treatment of a file contained in a multiple file tape environment is logically equivalent to the treatment of a sequential single-reel/unit file if the file is wholly contained on one reel, or to the treatment of a sequential multi-reel/unit file if the file is contained on more than one reel.

1. A CLOSE statement may only be executed for a file in an open mode. |However, when the file is an external file actually associated with SYSIN or with SYSOUT a CLOSE statement is accepted even though the file is not in an open mode; no action is then performed and the CLOSE is successful.|

2. For the purpose of showing the effect of various types of CLOSE statements as applied to various storage media, all files are divided into the following categories:

   a. Non-reel/unit. A file whose input or output medium is such that the concepts of rewind and reels/units have no meaning.

   b. Sequential single-reel/unit. A sequential file that is entirely contained on one reel/unit.

c. Sequential multi-reel/unit. A sequential file that is contained on more than one reel/unit.

d. Non-sequential single/multi-reel/unit. A file with organization other than sequential, which resides on a mass storage device.

3. The results of executing each type of CLOSE for each category of file are summarized in the table below.

Table 11-1. Relationship of File Categories and Formats of the CLOSE Statement

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

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

A. Effect on Previous Reels/Units

Input Files and Input-Output Files:

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

Output Files:

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

B. No Rewind of Current Reel

The current reel/unit is left in its current position.

C. Close File

Input Files and Input-Output Files (Sequential Access Mode):

If the file is positioned at its end and label records are specified for the file, the labels are processed according to the standard label convention. The behavior of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. If the file is positioned at its end and label records are not specified for the file, label processing does not take place. If the file is positioned other than at its end, the closing operations are executed, but there is no ending label processing.

Input Files and Input-Output Files (Random or Dynamic Access Mode); Output Files (Random, Dynamic, or Sequential Access Mode):

If label records are specified for the file, the labels are processed according to the standard label convention. The behavior of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. If label records are not specified for the file, label processing does not take place, but closing operations are executed.

D. Reel/Unit Removal

The current reel or unit is rewound when applicable, and the reel or unit is logically removed from the run unit; however, the reel or unit may be accessed again, in its proper order of reels or units within the file, if a CLOSE statement without the REEL or UNIT phrase is subsequently executed for this file followed by the execution of an OPEN statement for the file.

E. File Lock

This file is locked and cannot be opened again during this execution of this run unit.

F. Close Reel/Unit

Input Files and Input-Output Files:

The following operations take place:

1. If the current reel/unit is the last or only reel/unit for the file or the reel is a non reel/unit medium, there is no reel/unit swap and the current volume pointer remains unchanged.

2. If another reel/unit exists for the file, a reel/unit swap occurs, the current volume pointer is updated to point to the next reel/unit existing in the file and the standard beginning reel/unit label procedure is executed. If no data records exist for the current volume, another reel/unit swap occurs.

Output Files (Reel/Unit Media):

The following operations take place:

1. The standard ending reel/unit label procedure is executed.

2. A reel/unit swap. The current volume pointer is updated to point to the new reel/unit.

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

4. The next executed WRITE statement that references that file directs the next logical data record to the next reel/unit of the file.

   Output Files (Non-Reel/Unit Media):

   Execution of this statement is considered successful. The file remains in the open mode, and no action takes place except as specified in general rule 4.

G. Rewind

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

H. Optional Phrases Ignored

   The CLOSE statement is executed as if none of the optional phrases is present.

X. Illegal

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

4. The execution of the CLOSE statement causes the value of the I-O status associated with file-name-1 to be updated. (See "I-O Status" of the CLOSE statement.)

5. If an optional file is not present, no end-of-file or reel/unit processing is performed for the file and the file position indicator and the current volume pointer are unchanged.

6. Following the successful execution of a CLOSE statement without the REEL or UNIT phrase, the record area associated with file-name-1 is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

7. Following the successful execution of a CLOSE statement without the REEL or UNIT phrase, the file is removed from the open mode, and the file is no longer associated with the file connector.

8. If more than one file-name-1 is specified in a CLOSE statement, the result of executing this CLOSE statement is the same as if a separate CLOSE statement had been written for each file-name-1 in the same order as specified in the CLOSE statement.

9. The WITH NO REWIND and FOR REMOVAL phrases will have no effect at object time if they do not apply to the storage media on which the file resides.

11.8 COMPUTE

Description

The COMPUTE statement assigns to one or more data items the value of an arithmetic |or boolean| expression.

Format-1

```
COMPUTE {identifier-1 [ROUNDED]}...

                    {   =        }
                    { |--------| }
                    { | FROM   | } arithmetic-expression-1
                    { | EQUALS | }
                      |--------|

      [ON SIZE ERROR imperative-statement-1]

      [NOT ON SIZE ERROR imperative-statement-2]

      [END-COMPUTE]
```

Format-2

```
|-------------------------------------------------------------|
|                            {FROM  }                         |
| COMPUTE {identifier-2}... {=     } boolean-expression-1     |
|                            {EQUALS}                         |
|-------------------------------------------------------------|
```

Syntax Rules

1. Identifier-1 must reference either an elementary numeric item or an elementary numeric edited item.

|2. Identifier-2 must reference an elementary boolean data item.

3. The words FROM and EQUALS are equivalent to each other and to the symbol =. They may be used interchangeably and the choice is generally made for readability.|

General Rules

1. An arithmetic |or boolean| expression consisting of a single identifier or literal provides a method of setting the value of the data item referenced by identifier-1 |or identifier-2| equal to the literal or the value of the data item referenced by the single identifier.

2. If more than one identifier is specified for the result of the operation, that is preceding |FROM, EQUALS or| =, the value of the arithmetic |or boolean| expression is developed, and then this value is stored as the new value of each of the data items referenced by identifier-1 |or identifier-2|.

3. The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE. (see "Intermediate Data Item" in Chapter 10.)

4. Additional rules and explanations relative to this statement are given under the appropriate paragraphs. (See "Scope of Statements", "Intermediate Data Item", the "ROUNDED Phrase", the "Arithmetic Statements", "Overlapping Operands", the "SIZE ERROR Condition", and "Multiple Results in Arithmetic Statements" in Chapter 10.)

|5. The size of the hypothetical data item resulting from the evaluation of boolean-expression-1 is the size of the largest boolean item referenced in the expression. All rules referring to sending data items refer to this hypothetical data item and these sending data items are moved to the data item referenced by identifier-2 according to the rules for the MOVE statement.|

11.9 CONTINUE

Description

The CONTINUE statement is a no operation statement. It indicates that no executable statement is present.

Format

```
CONTINUE
```

Syntax Rules

The CONTINUE statement may be used anywhere a conditional statement or an imperative-statement may be used.

General Rules

The CONTINUE statement has no effect on the execution of the program.

11.10 DELETE

Description

The DELETE statement logically removes a record from a mass storage file.

Format

```
DELETE file-name-1 RECORD
     [INVALID KEY imperative-statement-1]
     [NOT INVALID KEY imperative-statement-2]
     [END-DELETE]
```

Syntax Rules

1. The INVALID KEY and the NOT INVALID KEY phrases 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 AFTER STANDARD EXCEPTION procedure is not specified.

General Rules

1. The file referenced by file-name-1 must be a mass storage file and must be open in the I-O mode at the time of the execution of this statement. (See the "OPEN Statement" in Chapter 12.)

2. For files in the sequential access mode, the last input-output statement executed for file-name-1 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 relative file in random or dynamic access mode, the Operating System logically removes from the file that record identified by the content of the relative key data item associated with file-name-1. If the file does not contain the record specified by the key, the invalid key condition exists. (See the "Invalid Key Condition" in Chapter 10.)

4. For an indexed file in random or dynamic access mode, the Operating System logically removes from the file the record identified by the content of the prime record key data item associated with the file-name-1. If the file does not contain the record specified by the key, the invalid key condition exists. (See the "Invalid Key Condition" in Chapter 10.)

5. After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.

6. The execution of a DELETE statement does not affect the content of the record area or the content of the data item referenced by the data-name specified in the DEPENDING ON phrase of the RECORD clause associated with file-name-1.

7. The file position indicator is not affected by the execution of a DELETE statement.

8. The execution of the DELETE statement causes the value of the I-O status associated with file-name-1 to be updated. (See "I-O Status" of the DELETE statement.)

9. Transfer of control following the successful or unsuccessful execution of the DELETE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the DELETE statement.

10. The END-DELETE phrase delimits the scope of the DELETE statement. A description of the function of the END-DELETE phrase is given in the appropriate paragraph. (See "Scope of Statements" in Chapter 10.)

11. If, during the execution of a DELETE statement with the NOT INVALID KEY phrase, an invalid key condition does not occur, control is transferred to imperative-statement-2 at the appropriate time as follows:

    a. If the execution of the DELETE statement is successful, after the record is deleted and after updating the I-O status associated with file-name-1.

    b. If the execution of the DELETE statement is unsuccessful for a reason other than an invalid key condition, after updating the I-O status associated with file-name-1, and after executing the procedure, if any, specified by a USE AFTER STANDARD EXCEPTION PROCEDURE statement applicable to file-name-1.

11.11 DISABLE

Description

The DISABLE statement notifies the message control system (MCS) to inhibit data transfer between specified output queues and destinations for output or between specified sources and input queues for input or between the program and one specified source or destination for input-output.

The WITH KEY phrase is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

Format

```
         {INPUT [TERMINAL]}                      {identifier-1}
DISABLE  {I-O TERMINAL    } cd-name-1 [WITH KEY  {            }]
         {OUTPUT          }                      {literal-1   }
```

Syntax Rules

1. Cd-name-1 must reference an input CD when the INPUT phrase is specified.

2. Cd-name-1 must reference an input-output CD when the I-O TERMINAL phrase is specified.

3. Cd-name-1 must reference an output CD when the OUTPUT phrase is specified.

4. Literal-1 or the content of the data item referenced by identifier-1 must be defined as alphanumeric.

General Rules

1. The DISABLE statement provides a logical disconnection between the MCS and the specified sources or destinations. When this logical disconnection is already in existence, or is to be handled by some other means external to this program, the DISABLE statement is not required in this program. No action is taken when a DISABLE statement is executed which specifies a source or destination which is already disconnected, except that the value in the STATUS KEY indicates this condition. The logical path for the transfer of data between the COBOL programs and the MCS is not affected by the DISABLE statement.

2. The MCS will insure that the execution of a DISABLE statement will cause the logical disconnection at the earliest time the source or destination is inactive. The execution of the DISABLE statement will never cause the remaining portion of the message to be terminated during transmission to or from a terminal.

3. When the INPUT phrase without the optional word TERMINAL is specified, the logical paths between the queue and sub-queues specified by the contents of data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name-1 and all the associated enabled sources are deactivated.

4. When the INPUT phrase with the optional word TERMINAL is specified, the logical paths between the source (as defined by the content of the data item referenced by data-name-7 (SYMBOLIC SOURCE)) and all of its associated queues and sub-queues are deactivated.

5. When the I-O TERMINAL phrase is specified, the logical path between the source (as defined by the content of the data item referenced by data-name-3 (SYMBOLIC TERMINAL)) and the program is deactivated.

6. When the OUTPUT phrase is specified, the logical paths are deactivated for all destinations specified by the content of each occurrence of data-name-5 up to and including the occurrence specified by the content of data-name-1 of the area referenced by cd-name-1.

7. Literal-1 or the content of the data item referenced by identifier-1 will be matched with a password built into the system. The DISABLE statement will be honored only if literal-1 or the content of the data item referenced by identifier-1 match the system password. When literal-1 or the content of the data item referenced by identifier-1 do not match the system password, the value of the STATUS KEY item in the area referenced by cd-name-1 is updated.

   The MCS is capable of handling a password of from one to ten characters inclusive.

11.12 DISPLAY

Description

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

Format

```
        |--------------------| {identifier-1}
DISPLAY | [WITH CONVERSION] | {             }...
        |--------------------| {literal-1    }

          {    mnemonic-name-1      }
          { |----------------------| }
          { |  SYSOUT              | }
    [UPON { | [ALTERNATE] CONSOLE  | }] [WITH NO ADVANCING]
          { |  ALTERNATE-CONSOLE   | }
          { |  TERMINAL            | }
            |----------------------|
```

Syntax Rules

1. If the mnemonic option is chosen, it can only be associated with the following device names: CONSOLE, ALTERNATE-CONSOLE, |ALTERNATE CONSOLE,| TERMINAL, SYSOUT, or corresponding suffixed device names in the SPECIAL-NAMES paragraph in the Environment Division.

2. If literal-1 is numeric, then it must be an unsigned integer.

General Rules

1. The DISPLAY statement causes the content of each operand to be transferred to the hardware device in the order listed.

|2. If WITH CONVERSION is specified, and identifier-1 references an elementary numeric data item, the data item is moved to a temporary data item of USAGE IS DISPLAY and of the same PICTURE as identifier-1 except it is always signed (with SIGN IS LEADING SEPARATE) if the PICTURE clause is not specified, the data is moved to a temporary data item of USAGE IS DISPLAY, in a legible form, according to its USAGE. The temporary item is then transferred to the hardware device in lieu of the data item referenced by identifier-1.|

   If WITH CONVERSION is omitted, then no conversion is applied between literal-1 or the data item referenced by identifier-1 and the hardware |device, except if mnemonic-name-1 refer to a suffixed device name. (See "Legible Equivalent" in Chapter 3). In this case, the contents of each operand are moved to its Legible Output Equivalent or set of concatenated Legible Output Equivalent. The Legible Output Equivalents are then transferred without conversion or alignment to the hardware device, in order listed, and without intervening separators or blanks.|

3. If mnemonic-name-1 is associated with SYSOUT, |or if SYSOUT is explicitly specified,| data is displayed upon the file whose internal-file-name is H_PR.

   If mnemonic-name-1 is associated with CONSOLE, |or if CONSOLE is explicitly specified,| data is displayed upon the main operator console.

   If mnemonic-name-1 is associated with ALTERNATE-CONSOLE or ALTERNATE CONSOLE, |or if ALTERNATE-CONSOLE or ALTERNATE CONSOLE are explicitly specified,| data is displayed upon the alternate operator console specified in the CONSOLE JCL statement. If no alternate console is specified, data is displayed upon the console from which the job is submitted.

   If mnemonic-name-1 is associated with TERMINAL, |or if TERMINAL is explicitly specified,| data is displayed upon the alternate operator console specified in the CONSOLE JCL statement. If no alternate console is specified, data is displayed upon the console from which the job is submitted. When the program interactively runs with an I0F terminal, data will be displayed upon that very terminal.

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

5. If the hardware device is capable of receiving data of the same size as the data item being transferred, then the data item is transferred.

6. If a hardware device is not capable of receiving data of the same size as the data item being transferred, then one of the following applies:

   a. If the size of the data item being transferred exceeds the size of the data that the hardware device is capable of receiving in a single transfer, the data beginning with the leftmost character is stored aligned to the left in the receiving hardware device, and additional data is requested.

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

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 without modifying the positioning of the hardware device between the successive operands.

8. If the UPON phrase is not specified, data is displayed upon the file whose internal-file-name is H_PR, i.e. the standard device is that implied by SYSOUT. |However, another default may be specified in the DISPLAY clause of the Default Section of the Control Division.|

9. If the WITH NO ADVANCING phrase is specified, then the positioning of the hardware device will not be reset to the next line or changed in any other way following the display of the last operand.

10. If the WITH NO ADVANCING phrase is not specified, then after the last operand has been transferred to the hardware device, the positioning of the hardware device will be reset to the leftmost position of the next line of the device.

11.13 DIVIDE

Description

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

Format 1

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

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-DIVIDE]
```

Format 2

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

              GIVING {identifier-3 [ROUNDED]}...

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-DIVIDE]
```

Format 3

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

              GIVING {identifier-3 [ROUNDED]}...

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-DIVIDE]
```

Format 4

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

           GIVING identifier-3 [ROUNDED] REMAINDER identifier-4

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-DIVIDE]
```

Format 5

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

           GIVING identifier-3 [ROUNDED] REMAINDER identifier-4

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-DIVIDE]
```

Syntax Rules

1. Each identifier must refer to an elementary numeric item, except that any identifier associated with the GIVING or REMAINDER phrase 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, which is the hypothetical data item resulting from the super-imposition of all receiving data items (except the REMAINDER data item) of a giving statement aligned on their decimal points, must not contain more than 18 digits |(up to 30 if the compiler is run with the LEVEL=NSTD parameter).|

General Rules

1. When Format 1 is used, literal-1 or the value of the data item referenced by identifier-1 is stored in a temporary data item. The value in the temporary data item is then divided into the value of the data item referenced by identifier-2. The value of the dividend (the data item referenced by identifier-2) is replaced by this quotient; similarly the temporary data item is divided into each successive occurrence of identifier-2 in the left-to-right order in which identifier-2 is specified.

2. When Format 2 is used, literal-1 or the value of the data item referenced by identifier-1 is divided into literal-2 or the value of the data item referenced by identifier-2 and the result is stored in each data item referenced by identifier-3.

3. When format 3 is used, literal-1 or the value of the data item referenced by identifier-1 is divided by literal-2 or the value of the data item referenced by identifier-2 and the result is stored in each data item referenced by identifier-3.

4. When Format 4 is used, literal-1 or the value of the data item referenced by identifier-1 is divided into literal-2 or the value of the data item referenced by identifier-2 and the result is stored in the data item referenced by identifier-3. The remainder is then calculated and the result is stored in the data item referenced by identifier-4. If identifier-4 is subscripted, then the subscript is evaluated immediately before the remainder is stored in the data item referenced by identifier-4.

5. When Format 5 is used, literal-1 or the value of the data item referenced by identifier-1 is divided by literal-2 or the value of the data item referenced by identifier-2 and the division operation continues as specified for format 4 above.

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

7. In Formats 4 and 5, the accuracy of the REMAINDER data item (identifier-4) is defined by the calculation described above.

   Appropriate decimal alignment and truncation (not rounding) will be performed for the value of the data item referenced by identifier-4, as needed.

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

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

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

9. Additional rules and explanations relative to this statement are given under the appropriate paragraphs. (See "Scope of Statements", "Intermediate Data Item", the "ROUNDED Phrase", the "Arithmetic Statements", "Overlapping Operands", the "SIZE ERROR Condition", and "Multiple Results in Arithmetic Statement" in Chapter 10. See also general rules 6 through 8 above for a discussion of the ROUNDED phrase and the SIZE ERROR phrase as they pertain to formats 4 and 5.)

11.14 ENABLE

Description

The ENABLE statement notifies the message control system (MCS) to allow data transfer between specified output queues and destinations for output or between specified sources and input queues for input or between the program and one specified source or destination for input-output.

The WITH KEY phrase is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

Format

```
        {INPUT [TERMINAL]}                      {identifier-1}
ENABLE  {I-O TERMINAL    } cd-name-1 [WITH KEY  {            }]
        {OUTPUT          }                      {literal-1   }
```

Syntax Rules

1. Cd-name-1 must reference an input CD when the INPUT phrase is specified.

2. Cd-name-1 must reference an input-output CD when the I-O TERMINAL phrase is specified.

3. Cd-name-1 must reference an output CD when the OUTPUT phrase is specified.

4. Literal-1 or the content of the data item referenced by identifier-1 must be defined as alphanumeric.

General Rules

1. The ENABLE statement provides a logical connection between the MCS and the specified sources or destinations. When this logical connection is already in existence, or is to be handled by some other means external to this program, the ENABLE statement is not required in this program. No action is taken when an ENABLE statement is executed which specifies a source or destination which is already connected, except that the value in the STATUS KEY indicates this condition. The logical path for the transfer of data between the COBOL programs and the MCS is not affected by the ENABLE statement.

2. When the INPUT phrase without the optional word TERMINAL is specified, the logical paths between the queue and sub-queues specified by the contents of data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name-1 and all the associated sources are activated.

3. When the INPUT phrase with the optional word TERMINAL is specified, the logical paths between the source (as defined by the content of the data item referenced by data-name-7 (SYMBOLIC SOURCE)) and all of its associated queues and sub-queues are activated.

4. When the I-O TERMINAL phrase is specified, the logical path between the source (as defined by the content of the data item referenced by data-name-3 (SYMBOLIC TERMINAL)) and the program is activated.

5. When the OUTPUT phrase is specified, the logical paths are activated for all destinations specified by the content of each occurrence of data-name-5 up to and including the occurrence specified by the content of data-name-1 of the area referenced by cd-name-1.

6. Literal-1 or the content of the data item referenced by identifier-1 will be matched with a password built into the system. The ENABLE statement will be honored only if literal-1 or the content of the data item referenced by identifier-1 match the system password. When literal-1 or the content of the data item referenced by identifier-1 do not match the system password, the value of the STATUS KEY item in the area referenced by cd-name-1 is updated.

   The MCS is capable of handling a password of from one to ten characters inclusive.

11.15 EVALUATE

Description

The EVALUATE statement describes a multi-branch, multi-join structure. It can cause multiple conditions to be evaluated. The subsequent action of the object program depends on the results of these evaluations.

Format

```
          {identifier-1}       {identifier-2}
          {literal-1   }       {literal-2   }
EVALUATE  {expression-1} [ALSO {expression-2}]...
          {TRUE        }       {TRUE        }
          {FALSE       }       {FALSE       }

        {    ANY                                        }
        {    condition-1                                }
        { |----------------------------|                 }
        { | [NOT] boolean-expression-1 |                 }
        { |----------------------------|                 }
        {    TRUE                                       }
{{WHEN  {    FALSE                                      }
        {          {identifier-3          }             }
        {   [NOT] {{literal-3             }             }
        {          {arithmetic-expression-1}             }
        {                                                }
        {         {THROUGH} {identifier-4          }  }  }
        {        [{       } {literal-4            }]}}
                  {THRU   } {arithmetic-expression-2}

        {    ANY                                        }
        {    condition-2                                }
        { |----------------------------|                 }
        { | [NOT] boolean-expression-2 |                 }
        { |----------------------------|                 }
        {    TRUE                                       }
[ALSO   {    FALSE                                      }]... }...
        {          {identifier-5          }             }
        {   [NOT] {{literal-5             }             }
        {          {arithmetic-expression-3}             }
        {                                                }
        {         {THROUGH} {identifier-6          }  }  }
        {        [{       } {literal-6            }]}}
                  {THRU   } {arithmetic-expression-4}

              imperative-statement-1}...

 [WHEN OTHER imperative-statement-2]

 [END-EVALUATE]
```

Syntax Rules

1. The operands or the words TRUE and FALSE which appear before the first WHEN phrase of the EVALUATE statement are referred to individually as selection subjects and collectively, for all those specified, as the set of selection subjects.

2. The operands or the words TRUE, FALSE, and ANY which appear in a WHEN phrase of an EVALUATE statement are referred to individually as selection objects and collectively, for all those specified in a single WHEN phrase, as the set of selection objects.

3. The words THROUGH and THRU are equivalent.

4. Two operands connected by a THROUGH phrase must be of the same class |and must not reference boolean data items.| The two operands thus connected constitute a single selection object.

5. The number of selection objects within each set of selection objects must be equal to the number of selection subjects.

6. Each selection object within a set of selection objects must correspond to the selection subject having the same ordinal position within the set of selection subjects according to the following rules:

    a. Identifiers, literals, arithmetic expressions |, or boolean expressions| appearing within a selection object must be valid operands for comparison to the corresponding operand in the set of selection subjects. (See "Relation Condition" in Chapter 10.)

    b. Condition-1, condition-2 or the words TRUE or FALSE appearing as a selection object must correspond to a conditional expression or the words TRUE or FALSE in the set of selection subjects.

    c. The word ANY may correspond to a selection subject of any type.

General Rules

1. The execution of the EVALUATE statement operates as if each selection subject and selection object were evaluated and assigned a numeric non-numeric |or boolean| value, a range of numeric or non-numeric values, or a truth value. These values are determined as follows:

    a. Any selection subject specified by identifier-1, identifier-2 and any selection object specified by identifier-3, identifier-5, without either the NOT or the THROUGH phrases, are assigned the value and class of the data item referenced by the identifier.

    b. Any selection subject specified by literal-1, literal-2 and any selection object specified by literal-3, literal-5, without either the NOT or the THROUGH phrases, are assigned the value and class of the specified literal. If literal-3, literal-5 is the figurative constant ZERO, it is assigned the class of the corresponding selection subject.

    c. Any selection subject in which expression-1, expression-2, is specified as an arithmetic expression and any selection object, without either the NOT or the THROUGH phrases, in which arithmetic-expression-1, arithmetic-expression-3, is specified are assigned a numeric value according to the rules for evaluating an arithmetic expression. (See "Arithmetic Expressions" in Chapter 10.)

    |d. Any selection subject in which expression-1, expression-2 is specified as a boolean expression and any selection object without the NOT phrase in which boolean-expression-1, boolean-expression-2 is specified are assigned a boolean value according to the rules for evaluating boolean expressions (See "Boolean Expressions" in Chapter 10).|

    e. Any selection subject in which expression-1, expression-2 is specified as a conditional expression and any selection object in which condition-1, condition-2 is specified are assigned a truth value according to the rules for evaluating conditional expressions. (See "Conditional Expressions" in Chapter 10.)

    f. Any selection subject or any selection object specified by the words TRUE or FALSE is assigned a truth value. The truth value 'true' is assigned to those items specified with the word TRUE, and the truth value 'false' is assigned to those items specified with the word FALSE.

    g. Any selection object specified by the word ANY is not further evaluated.

    h. If the THROUGH phrase is specified for a selection object, without the NOT phrase, the range of values includes all permissible values of the selection subject that are greater than or equal to the first operand and less than or equal to the second operand according to the rules for comparison. (see "Relation Condition" in Chapter 10.)

    i. If the NOT phrase is specified for a selection object, the values assigned to that item are all permissible values of the selection subject not equal to the value, or not included in the range of values, that would have been assigned to the item had the NOT phrase not been specified.

2. The execution of the EVALUATE statement then proceeds as if the values assigned to the selection subjects and selection objects were compared to determine if any WHEN phrase satisfies the set of selection subjects. This comparison proceeds as follows:

   a. Each selection object within the set of selection objects for the first WHEN phrase is compared to the selection subject having the same ordinal position within the set of selection subjects. One of the following conditions must be satisfied if the comparison is to be satisfied:

      (1) If the items being compared are assigned numeric, non-numeric |pointer or boolean| values, or a range of numeric or non-numeric values, the comparison is satisfied if the value, or one of the range of values, assigned to the selection object is equal to the value assigned to the selection subject according to the rules for comparison. (See "Relation Condition" in Chapter 10.)

      (2) If the items being compared are assigned truth values, the comparison is satisfied if the items are assigned the identical truth value.

      (3) If the selection object being compared is specified by the word ANY, the comparison is always satisfied regardless of the value of the selection subject.

   b. If the above comparison is satisfied for every selection object within the set of selection objects being compared, the WHEN phrase containing that set of selection objects is selected as the one satisfying the set of selection subjects.

   c. If the above comparison is not satisfied for one or more selection object within the set of selection objects being compared, that set of selection objects does not satisfy the set of selection subjects.

   d. This procedure is repeated for subsequent sets of selection objects, in the order of their appearance in the source program, until either a WHEN phrase satisfying the set of selection subjects is selected or until all sets of selection objects are exhausted.

3. After the comparison operation is completed, execution of the EVALUATE statement proceeds as follows:

   a. If a WHEN phrase is selected, execution continues with the first imperative-statement-1 following the selected WHEN phrase.

   b. If no WHEN phrase is selected and a WHEN OTHER phrase is specified, execution continues with imperative-statement-2.

   c. The scope of execution of the EVALUATE statement is terminated when execution reaches the end of imperative-statement-1 of the selected WHEN phrase or the end of imperative-statement-2, or when no WHEN phrase is selected an no WHEN OTHER phrase is specified. (See "Scope of Statements" in Chapter 10.)

11.16 EXAMINE

Description

|The EXAMINE statement counts and/or replaces occurrence of a given character in a data item.|

Format

```
|-------------------------------------------------------------------|
|                                                                   |
|EXAMINE identifier-5                                               |
|-------                                                            |
|               {ALL        } {literal-1   }                       |
|    {TALLYING {LEADING    } {            }                     }|
|    {         {UNTIL FIRST} {identifier-1}                     }|
|    {                                                          }|
|    {                   {literal-2   }                         }|
|    {    [REPLACING BY {            }]                         }|
|    {                   {identifier-2}                         }|
|    {                                                          }|
|    {          { ALL         } {literal-3   }    {literal-4   }}|
|    {REPLACING { LEADING     } {            } BY {            }}|
|    {          {[UNTIL] FIRST} {identifier-3}    {identifier-4}}|
|                                                                   |
|-------------------------------------------------------------------|
```

Syntax Rules

|1. Identifier-5 must have a usage which is DISPLAY, explicitly or implicitly.

2. Each literal and every identifier except identifier-5 must name a single character or single-character data item, respectively, belonging to a class consistent with that of identifier-5. Also, each literal may be any figurative constant except ALL (the figurative constant is understood to refer to a single instance of that constant's value).

3. Signed numeric literals are not permitted in the EXAMINE statement.|

General Rules

|1. Identifier-5 is examined, character by character, from left to right, whether identifier-5 is numeric or non-numeric. However, if it is numeric and has an operational sign (as indicated by an S in its PICTURE character-string), its sign is ignored by the EXAMINE statement.

2. Execution of the EXAMINE statement with the TALLYING option creates an integral count that is placed in the special register TALLY. The value of this count depends on the TALLYING option used:

   a. When ALL is used, the number of occurrences in identifier-5 of literal-1 or identifier-1 is counted.

   b. When LEADING is used, the count is equal to the number of occurrences in identifier-5 of literal-1 or identifier-1, before the first occurrence of a character other than literal-1 or identifier-1, or the right-hand boundary is encountered.

   c. When UNTIL FIRST is used, the count is equal to the number of characters in identifier-5 that are not equal to literal-1 or identifier-1, encountered before the first occurrence of literal-1 or identifier-1, or the right-hand boundary.

3. The word TALLY (the special TALLY register), may be used as a data-name wherever an integer elementary data item may appear.

4. When either REPLACING option is used, replacement rules are as follows, subject to General Rule 2 above:

   a. When ALL is used then the item following BY (literal-2, identifier-2 or literal-4, identifier-4) is substituted for each occurrence of the item following ALL (literal-1, identifier-1 or literal-3, identifier-3), respectively.

   b. When LEADING is used, the substitution of the item following BY (literal-2, identifier-2 or literal-4, identifier-4) terminates as soon as a character other than the item following LEADING (literal-1, identifier-1 or literal-3, identifier-3) or the right-hand boundary of the data item is encountered.

   c. When UNTIL FIRST is used, the substitution of the item following BY (literal-2, identifier-2 or literal-4, identifier-4) terminates as soon as the item following UNTIL FIRST (literal-1, identifier-1 or literal-3, identifier-3) or the right-hand boundary of the data item is encountered.

   d. When FIRST is used, the first occurrence of the item following FIRST (literal-1, identifier-1 or literal-3, identifier-3) is replaced by the item following BY (literal-2, identifier-2 or literal-4, identifier-4).|

11.17 EXIT

Description

The EXIT statement provides a common end point for a series of procedures, or marks the logical end of a called program.

Format

```
                    |-----------------------|
EXIT [ PROGRAM | [GIVING identifier-1] | ].
                    |-----------------------|
```

Syntax Rules

1. The EXIT statement without the PROGRAM phrase must appear only in a sentence by itself and comprise the only sentence in the paragraph.

2. If an EXIT PROGRAM statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

3. The EXIT PROGRAM statement must not appear in a declarative procedure in which the GLOBAL phrase is specified.

4. |Identifier-1 must reference either:

   a. an elementary data item of USAGE COMP-1, COMP-2, COMP-9, COMP-10 or pointer,

   b. an elementary data item of USAGE ALPHABETIC or ALPHANUMERIC of one character in length.|

General Rules

1. An EXIT statement without the optional phrase PROGRAM serves only to enable the user to assign a procedure-name to a given point in a program. Such an EXIT statement has no other effect on the compilation or execution of the program.

2. If the EXIT PROGRAM statement is executed in a program which is not under the control of a calling program, the EXIT PROGRAM statement causes execution of the program to continue with the next executable statement.

3. The execution of an EXIT PROGRAM statement in a called program which does not possess the initial attribute causes execution to continue with the next executable statement following the CALL statement in the calling program. The program state of the calling program is not altered and is identical to that which existed at the time it executed the CALL statement except that the contents of data items and the contents of data files shared between the calling and called program may have been changed. The program state of the called program is not altered except that the ends of the ranges of all PERFORM statements executed by that called program are considered to have been reached.

4. Besides the actions specified in general rule 3, the execution of an EXIT PROGRAM statement in a called program which possesses the initial attribute is equivalent also to executing a CANCEL statement referencing that program. (See the "CANCEL Statement" in Chapter 11.)

5. |The GIVING phrase is used when the calling program is written in C Language and needs a return value. The content of the data item referenced by identifier-1 is made available to the calling program after control is transferred to the calling program.|

11.18 GENERATE

Description

The GENERATE statement directs the Report Writer Control System (RWCS) to produce a report in accordance with the Report Description that was specified in the Report Section of the Data Division.

Format

```
          {data-name-1  }
GENERATE  {             }
          {report-name-1}
```

Syntax Rules

1. Data-name-1 must name a TYPE DETAIL report group and may be qualified by a report name.

2. Report-name-1 may be used only if the referenced Report Description contains:

   a. A CONTROL clause, and

   b. Not more than one DETAIL report group, and

   c. At least one body group.

General Rules

1. In response to a GENERATE report-name-1 statement, the RWCS performs summary processing. If all of the GENERATE statements that are executed for a report are of the form GENERATE report-name-1, then the report that is produced is called a summary report. A summary report is one in which no DETAIL report group is presented.

2. In response to a GENERATE data-name-1 statement, the RWCS performs detail processing that includes certain processing that is specific for the DETAIL report group designated by the GENERATE statement. Normally, the execution of a GENERATE data-name-1 statement causes the RWCS to present the designated DETAIL report group.

3. During the execution of the chronologically first GENERATE statement for a given report, the RWCS saves the values within the control data items. During the execution of the second and subsequent GENERATE statements for the same report, and until a control break is detected, the RWCS utilizes this set of control values to determine whether a control break has occurred. When a control break occurs, the RWCS saves the new set of control values, which it thereafter uses to sense for a control break until another control break occurs.

4. During report presentation, an automatic function of the RWCS is to process PAGE HEADING and PAGE FOOTING report groups, if defined, when the RWCS must advance the report to a new page for the purpose of presenting a body group (see "Presentation Rules Tables", Chapter 8).

5. When the chronologically first GENERATE statement for a given report is executed, the RWCS processes, in order, the report groups that are named below, provided that such report groups are defined within the Report Description. The RWCS also processes PAGE HEADING and PAGE FOOTING report groups as described in General Rule 4. The actions taken by the RWCS when it processes each type of report group are explained under the appropriate paragraph (see "TYPE Clause", Chapter 9).

    a. The REPORT HEADING report group is processed.

    b. The PAGE HEADING report group is processed.

    c. All CONTROL HEADING report groups are processed from major to minor.

    d. If a GENERATE data-name-1 statement is being executed, the processing for the designated DETAIL report group is performed. If a GENERATE report-name-1 statement is being executed certain of the steps that are involved in the processing of a DETAIL report group are performed (See the "TYPE Clause", Chapter 9).

6. When a GENERATE statement other than the chronologically first is executed for a given report, the RWCS performs the steps enumerated below, as applicable. The RWCS also processes PAGE HEADING and PAGE FOOTING report groups as described in General Rule 4. The actions taken by the RWCS when it processes each type of report group are explained under the appropriate paragraph (see the "TYPE Clause", Chapter 9).

    a. Sense for control break. The rules for determining the equality of control data items are the same as those specified for relation conditions. If a control break has occurred then:

        (1) Enable the CONTROL FOOTING USE procedures and CONTROL FOOTING SOURCE clauses to access the control data item values that the RWCS used to detect a given control break (see the "TYPE Clause", Chapter 9).

        (2) Process the CONTROL FOOTING report groups in the order minor to major. Only CONTROL FOOTING report groups that are not more major than the highest level at which a control break occurred are processed.

        (3) Process the CONTROL HEADING report groups in the order major to minor. Only the CONTROL HEADING report groups that are not more major than the highest level at which a control break occurred are processed.

    b. If a GENERATE data-name-1 statement is being executed, the processing for the designated DETAIL report group is performed. If a GENERATE report-name-1 statement is being executed, certain of the steps that are involved in the processing of a DETAIL report group are performed (see the "TYPE Clause", Chapter 9).

7. GENERATE statements for a report can be executed only after an INITIATE statement for the report has been executed and before a TERMINATE statement for the report has been executed.

11.19 GO TO

Description

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

The optionality of procedure-name-1 in format 1 of the GO TO statement is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

Format 1

```
GO TO [procedure-name-1]
```

Format 2

```
GO TO {procedure-name-1}... DEPENDING ON identifier-1
```

Syntax Rules

1. Identifier-1 must reference a numeric elementary data item which is an integer.

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. A format 1 GO TO statement, without procedure-name-1, can only appear in a single statement paragraph.

4. If a GO TO statement represented by format 1 appears in a consecutive sequence of imperative statements within a sentence, it must appear 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. If procedure-name-1 is not specified in a GO TO statement represented by format 1, an ALTER statement referring to this GO TO statement must be executed prior to the execution of this GO TO statement.

3. When a GO TO statement represented by format 2 is executed, control is transferred to procedure-name-1, etc., depending on the value of identifier-1 being 1, 2 ..., n. If the value of identifier-1 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.

12. Procedure Division - Statements (IF to REWRITE)

This chapter describes the statements from IF to REWRITE (inclusive).

The statements concerned are as follows:

- IF
- INITIALIZE
- INITIATE
- INSPECT
- MERGE
- MOVE
- MULTIPLY
- OPEN
- PERFORM
- PURGE
- READ
- RECEIVE
- RELEASE
- RETURN
- REWRITE

12.1 IF

Description

The IF statement causes a condition (See "Conditional Expressions" in Chapter 10) to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false.

Format

```
                     {{statement-1}...}
IF condition-1 THEN  {                 }
                     { NEXT SENTENCE   }

            {ELSE {statement-2}... [END-IF]}
            {ELSE NEXT SENTENCE            }
            {END-IF                        }
```

Syntax Rules

1. Statement-1 and statement-2 represent either an imperative statement or a conditional statement optionally preceded by an imperative statement. A further description of the rules governing statement-1 and statement-2 is given elsewhere. (See "Scope of Statements" in Chapter 10.)

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

3. If the END-IF phrase is specified, the NEXT SENTENCE phrase must not be specified.

General Rules

1. The scope of an IF statement may be terminated by any of the following:

   a. An END-IF phrase at the same level of nesting.

   b. A separator period.

   c. If nested, by an ELSE phrase associated with an IF statement at a higher level of nesting. (See "Scope of Statements" in Chapter 10.)

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

   a. If the condition is true and statement-1 is specified, control is transferred to the first statement of statement-1 and execution continues according to the rules for each statement specified in statement-1. If a procedure branching or conditional statement is executed which causes an explicit transfer of control, control is explicitly transferred in accordance with the rules of that statement. Upon completion of the execution of statement-1, the ELSE phrase, if specified, is ignored and control passes to the end of the IF statement.

   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 and statement-2 is specified, statement-1 or its surrogate NEXT SENTENCE is ignored, control is transferred to the first statement of statement-2, and execution continues according the rules for each statement specified in statement-2. If a procedure branching or conditional statement is executed which causes an explicit transfer of control, control is explicitly transferred in accordance with the rules of that statement. Upon completion of the execution of statement-2, control passes to the end of the IF statement.

   d. If the condition is false and the ELSE phrase is not specified, statement-1 is ignored and control passes to the end of the IF statement.

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

3. Statement-1 and/or statement-2 may contain an IF statement. In this case, the IF statement is said to be nested. More detailed rules on nesting are given in the appropriate paragraph. (See "Scope of Statements" in Chapter 10.)

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

12.2 INITIALIZE

Description

The INITIALIZE statement provides the ability to set selected types of data fields to predetermined values, e.g., numeric data to zeroes or alphanumeric data to spaces.

Format

```
INITIALIZE {identifier-1}...

                {   ALPHABETIC          }
                { |---------|           }
                { |  BOOLEAN  |         }           {identifier-2}
   [REPLACING   { |---------|           } DATA BY {             }... ]
                {   ALPHANUMERIC        }           {literal-1    }
                {   NUMERIC             }
                {   ALPHANUMERIC-EDITED }
                {   NUMERIC-EDITED      }
```

Syntax Rules

1. Literal-1 and the data item referenced by identifier-2 represent the sending area; the data item referenced by identifier-1 represents the receiving area.

2. Each category stated in the REPLACING phrase must be a permissible category as a receiving operand in a MOVE statement where the corresponding data item referenced by identifier-2 or literal-1 is used as the sending operand. (See the "MOVE Statement" in Chapter 12.)

3. The same category cannot be repeated in a REPLACING phrase.

4. The description of the data item referenced by identifier-1 and any items subordinate to identifier-1 may not contain the DEPENDING phrase of the OCCURS clause.

5. An index |or pointer| data item may not appear as an operand of an INITIALIZE statement.

|6. If the data item referenced by identifier-1 contains a RENAMES clause:

   a. If that item is elementary, it is initialized according to its category;

   b. If that item is a group item, it is initialized as if it was an elementary item of the category alphanumeric.|

General Rules

1. The key word following the word REPLACING corresponds to a category of data as defined elsewhere in this document. (See "Concept of Classes of Data" in Chapter 3.)

2. Whether identifier-1 references an elementary item or a group item, all operations are performed as if a series of MOVE statements had been written, each of which has an elementary item as its receiving field, subject to the following rules:

   If the REPLACING phrase is specified and if identifier-1 references a group item, any elementary item within the data item referenced by identifier-1 is initialized only if it belongs to the category specified in the REPLACING phrase.

   If the REPLACING phrase is specified and if identifier-1 references an elementary item, that item is initialized only if it belongs to the category specified in the REPLACING phrase.

   This initialization takes place as follows: The data item referenced by identifier-2 or literal-1 acts as the sending operand in an implicit MOVE statement to the identified item.

   All such elementary receiving fields, including all occurrences of table items within the group, are affected; the only exceptions are those fields specified in general rules 3 and 4.

3. Index |or pointer| data items and elementary FILLER data items are not affected by the execution of an INITIALIZE statement.

4. Any item that is subordinate to a receiving area identifier and which contains the REDEFINES clause or any item that is subordinate to such an item is excluded from this operation. However, a receiving area identifier may itself have a REDEFINES clause or be subordinate to a data item with a REDEFINES clause.

5. When the statement is written without the REPLACING phrase, data items of the categories alphabetic, alphanumeric and alphanumeric edited are set to spaces; data items of the categories numeric, numeric edited |and boolean| are set to zeros. In this case, the operation is as if each affected data item is the receiving area in an elementary MOVE statement with the indicated source literal (i.e., spaces or zeros).

6. In all cases, the content of the data item referenced by identifier-1, etc., is set to the indicated value in the order (left to right) of the appearance of identifier-1, etc., in INITIALIZE statement. Within this sequence, where identifier-1, etc., references a group item, affected elementary items are initialized in the sequence of their definition within the group.

7. If identifier-1, etc., occupy the same storage area as identifier-2, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See "Overlapping Operands" in Chapter 10.)

12.3 INITIATE

Description

The INITIATE statement causes the Report Writer Control System (RWCS) to begin the processing of a report.

Format

```
INITIATE {report-name-1}...
```

Syntax Rule

Each report-name-1 must be defined by a Report Description entry in the Report Section of the Data Division.

General Rules

1. The INITIATE statement performs the following initialization functions for each named report:

   a. All sum counters are set to zero

   b. LINE-COUNTER is set to zero

   c. PAGE-COUNTER is set to one (1).

2. The INITIATE statement does not place the file to which the report is assigned in the open mode; therefore, an OPEN statement with either the OUTPUT phrase or the EXTEND phrase for the file must be executed prior to the execution of the INITIATE statement.

3. A subsequent INITIATE statement for a particular report-name-1 must not be executed unless an intervening TERMINATE statement has been executed for that report-name-1.

4. If more than one report-name-1 is specified in an INITIATE statement, the result of executing this INITIATE statement is the same as if a separate INITIATE statement had been written for each report name in the same order as specified in the INITIATE statement.

12.4 INSPECT

Description

The INSPECT statement tallies or replaces occurrences of single characters or groups of characters in a data item.

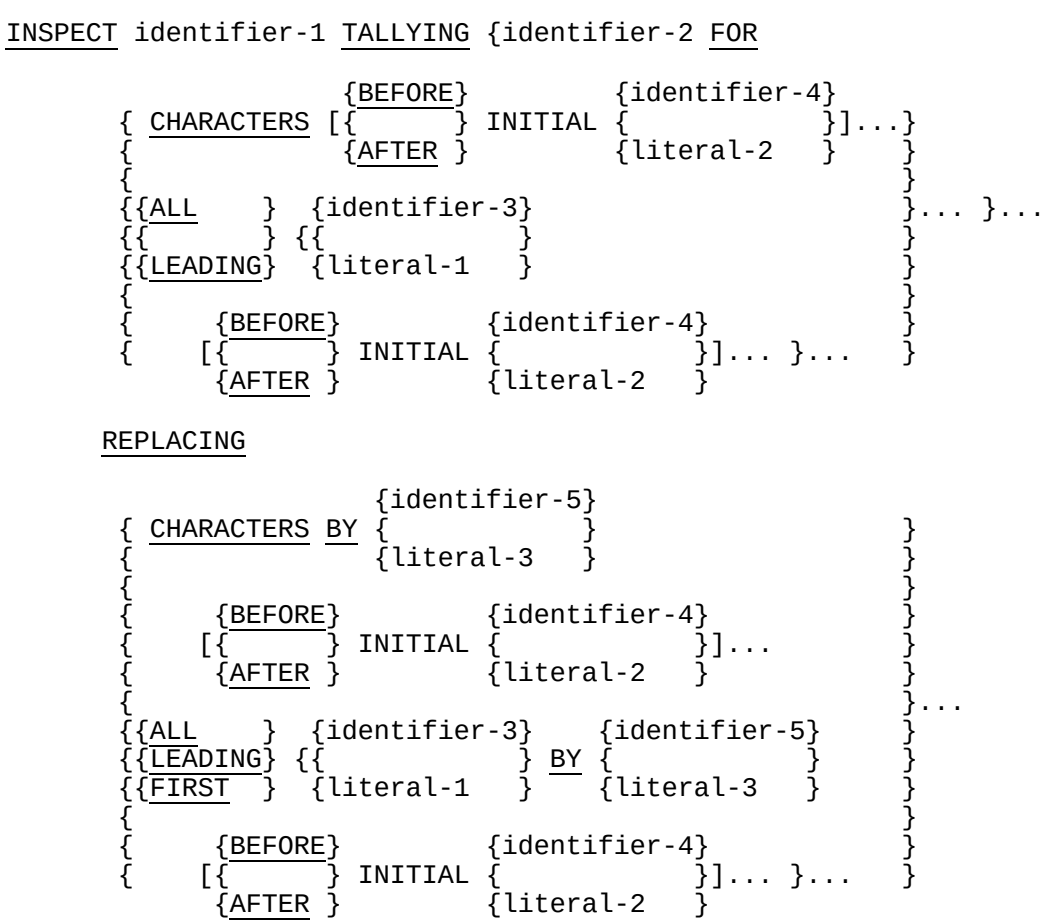
Format 1

```
INSPECT identifier-1 TALLYING {identifier-2 FOR

                 {BEFORE}         {identifier-4}
   { CHARACTERS [{      } INITIAL {            }]...}
   {             {AFTER }         {literal-2   }    }
   {                                                 }
   {{ALL    }  {identifier-3}                       }... }...
   {{       } {{            }                       }
   {{LEADING}  {literal-1   }                       }
   {                                                 }
   {     {BEFORE}         {identifier-4}            }
   {    [{      } INITIAL {            }]... }...   }
         {AFTER }         {literal-2   }
```

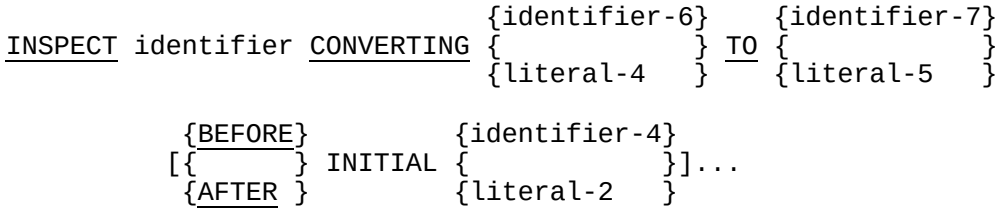
Format 2

```
INSPECT identifier-1 REPLACING

                  {identifier-5}
   { CHARACTERS BY {            }                  }
   {               {literal-3   }                  }
   {                                               }
   {     {BEFORE}         {identifier-4}           }
   {    [{      } INITIAL {            }]..  .     }
   {     {AFTER }         {literal-2   }           }
   {                                               }...
   {{ALL    }  {identifier-3}    {identifier-5}     }
   {{LEADING} {{            } BY {            }     }
   {{FIRST  }  {literal-1   }    {literal-3   }     }
   {                                               }
   {     {BEFORE}         {identifier-4}           }
   {    [{      } INITIAL {            }]... }..  . }
         {AFTER }         {literal-2   }
```

Format 3



Format 4



Syntax Rules

All Formats

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

2. Identifier-3, ... identifier-n must reference an elementary item described, implicitly or explicitly.as USAGE IS DISPLAY.

3. Each literal must be non-numeric literal and must not be a figurative constant that begins with the word ALL. If literal-1, literal-2, or literal-4 is a figurative constant, it refers to an implicit one character data item.

4. No more than one BEFORE phrase and one AFTER phrase can be specified for any one ALL, LEADING, CHARACTERS, FIRST or CONVERTING phrase.

Formats 1 and 3 Only

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

Formats 2 and 3 Only

6. The size of literal-3 or the data item referenced by identifier-5 must be equal to the size of literal-1 or the data item referenced by identifier-3. When a figurative constant is used as literal-3, the size of the figurative constant is equal to the size of literal-1 or the size of the data item referenced by identifier-3.

7. When the CHARACTERS phrase is used, literal-2, literal-3 or the size of the data item referenced by identifier-4, identifier-5 must be one character in length.

Format 4

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

9. The same character must not appear more than once either in literal-4 or in the data item referenced by identifier-6.

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 5 through 7.

2. For use in the INSPECT statement, the content 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 reference an alphabetic or alphanumeric data item, 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 reference an alphanumeric edited, numeric edited, unsigned numeric or |boolean| data items, the data item is inspected as though it had been re-defined as alphanumeric (see "General Rule" 2a) and the INSPECT statement had been written to reference the re-defined data item.

    c. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 reference a signed numeric data item, the data item is inspected as though it had been moved to an unsigned numeric data item with length equal to the length of the signed item excluding any separate sign position, and then the rules in general rule 2b had been applied. (See the "MOVE Statement" in Chapter 12.) If identifier-1 is a signed numeric item, the original value of the sign is retained upon completion of the INSPECT statement.

3. In general rules 5 through 17 all references to literal-1, literal-2, literal-3, literal-4 or literal-5 apply equally to the content of the data item referenced by identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7, respectively.

4. If any identifier is subscripted or is a function-identifier, the subscript or function-identifier is evaluated only once as the first operation in the execution of the INSPECT statement.

Formats 1 and 2 Only

5. During inspection of the content of the data item referenced by identifier-1, each properly matched occurrence of literal-1 is tallied (format 1) or replaced by literal-3 (format 2).

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

   a. The operands of the TALLYING or REPLACING phrase are considered in the order they are specified in the INSPECT statement from left to right. The first literal-1 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 matches that portion of the content of the data item referenced by identifier-1 if they are equal, character for character and:

      (1) If neither LEADING nor FIRST is specified; or

      (2) If the LEADING adjective applies to literal-1 and literal-1 is a leading occurrence as defined in general rules 10 and 13; or

      (3) If the FIRST adjective applies to literal-1 and literal-1 is the first occurrence as defined in general rule 13.

   b. If no match occurs in the comparison of the first literal-1 the comparison is repeated with each successive literal-1, if any, until either a match is found or there is no next successive literal-1. When there is no next successive literal-1, 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.

   c. Whenever a match occurs, tallying or replacing takes place as described in general rules 10 and 13. 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.

   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 6a through 6d above as if it had been specified by literal-1, except that no comparison to the content of the data item referenced by identifier-1 takes place. This implied character is considered always to match the leftmost character of the content of the data item referenced by identifier-1 participating in the current comparison cycle.

7. The comparison operation defined in general rule 6 is restricted by the BEFORE and AFTER phrase as follows:

   a. If neither the BEFORE or AFTER phrase is specified, literal-1 or the implied operand of the CHARACTERS phrase participates in the comparison operation as described in general rule 6. Literal-1 or the implied operand of the CHARACTERS phrase is first eligible to participate in matching at the leftmost position of identifier-1.

   b. If the BEFORE phrase is specified, the associated literal-1 or the implied operand of the CHARACTERS phrase participates only in those comparison cycles which involve that portion of the content of the data item referenced by identifier-1 from its leftmost character position up to, but not including, the first occurrence of literal-2 within the content 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 6 is begun. If, on any comparison cycle, literal-1 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the content of the data item referenced by identifier-1. If there is no occurrence of literal-2 within the content of the data item referenced by identifier-1, its associated literal-1 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 or the implied operand of the CHARACTERS phrase participate only in those comparison cycles which involve that portion of the content 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 within the content of the data item referenced by identifier-1 to the rightmost character position of the data item referenced by identifier-1. This is the character position at which literal-1 or the implied operand of the CHARACTERS phrase is first eligible to participate in matching. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 6 is begun. If, on any comparison cycle, literal-1 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the content of the data item referenced by identifier-1. If there is no occurrence of literal-2 within the content of the data item referenced by identifier-1, its associated literal-1 or the implied operand of the CHARACTERS phrase is never eligible to participate in the comparison operation.

Format 1

8. The required words ALL and LEADING are adjectives that apply to each succeeding literal-1 or identifier-3 until the next adjective appears.

9. The content of the data item referenced by identifier-2 are not initialized by the execution of the INSPECT statement.

10. The rules for tallying are as follows:

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

    b. If the LEADING phrase is specified, the content of the data item referenced by identifier-2 is incremented by one for the first and each subsequent contiguous occurrence of literal-1 matched within the content 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 content of the data item referenced by identifier-2 is incremented by one for each character matched, in the sense of general rule 6e, within the content of the data item referenced by identifier-1.

11. If identifier-1, identifier-3, or identifier-4 occupies the same storage area as identifier-2, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See "Overlapping Operands" in Chapter 10.)

Format 2

12. The required words ALL, LEADING, and FIRST are adjectives that apply to each succeeding BY phrase until the next adjective appears.

13. The rules for replacement are as follows:

    a. When the CHARACTERS phrase is specified, each character matched, in the sense of general rule 6e, in the content of the data item referenced by identifier-1 is replaced by literal-3.

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

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

    d. When the adjective FIRST is specified, the leftmost occurrence of literal-1 matched within the content of the data item referenced by identifier-1 is replaced by literal-3. This rule applies to each successive specification of the FIRST phrase regardless of the value of literal-1.

14. If identifier-3, identifier-4 or identifier-5 occupies the same storage area as identifier-1, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See "Overlapping Operands" in Chapter 10.)

Format 3

15. 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.Subscripting associated with any identifier in the format 2 statement is evaluated only once before executing the format 1 statement.

Format 4

16. A format 4 INSPECT statement is interpreted and executed as though a format 2 INSPECT statement specifying the same identifier-1 had been written with a series of ALL phrases, one for each character of literal-4. The effect is as if each of these ALL phrases referenced, as literal-1, a single character of literal-4 and referenced, as literal-3, the corresponding single character of literal-5. Correspondence between the characters of literal-4 and the characters of literal-5 is by ordinal position within the data item.

17. If identifier-4, identifier-6, or identifier-7 occupies the same storage area as identifier-1, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See "Overlapping Operands" in Chapter 10.)

Examples:

In each of the following examples of the INSPECT statement, COUNT-n is assumed to be zero immediately prior to execution of the statement. The results shown for each example, except the last, are the result of executing the two successive INSPECT statements shown above them.

Example 1:

```
INSPECT ITEM TALLYING
  COUNT-0 FOR ALL "AB", ALL "D"
  COUNT-1 FOR ALL "BC"
  COUNT-2 FOR LEADING "EF"
  COUNT-3 FOR LEADING "B"
  COUNT-4 FOR CHARACTERS;

INSPECT ITEM REPLACING
  ALL "AB" BY "XY", "D" BY "X"
  ALL "BC" BY "VW"
  LEADING "EF" BY "TU"
  LEADING "B" BY "S"
  FIRST "G" BY "R"
  FIRST "G" BY "P"
  CHARACTERS BY "Z"
```

```
Initial Value                                          Final Value
 of          COUNT-0 COUNT-1 COUNT-2 COUNT-3 COUNT-4     of
ITEM                                                    ITEM

EFABDBCGABEFGG 3       1       1       0       5    TUXYXVWRXYZZPZ

BABABC         2       0       0       1       1    SXYXYZ

BBBC           0       1       0       2       0    SSVW
```

Example 2:

```
INSPECT ITEM TALLYING
  COUNT-0 FOR CHARACTERS
  COUNT-1 FOR ALL "A";

INSPECT ITEM REPLACING
  CHARACTERS BY "Z" ALL "A" BY "X"

Initial Value                              Final Value
 of            COUNT-0      COUNT-1        of
ITEM                                       ITEM

BBB               3            0          ZZZ

ABA               3            0          ZZZ
```

Example 3:

```
INSPECT ITEM TALLYING
  COUNT-0 FOR ALL "AB" BEFORE "BC"
  COUNT-1 FOR LEADING "B" AFTER "D"
  COUNT-2 FOR CHARACTERS AFTER "A" BEFORE "C";

INSPECT ITEM REPLACING
  ALL "AB" BY "XY" BEFORE "BC"
  LEADING "B" BY "W" AFTER "D"
  FIRST "E" BY "V" AFTER "D"
  CHARACTERS BY "Z" AFTER "A" BEFORE "C"

Initial Value                                  Final Value
 of               COUNT-0 COUNT-1  COUNT-2        of
ITEM                                           ITEM

BBEABDABABBCABEE    3        0        2       BBEXYZXYXYZCABVE

ADDDDC              0        0        4       AZZZZC

ADDDDA              0        0        5       AZZZZZ

CDDDDC              0        0        0       CDDDDC

BDBBBDB             0        3        0       BDWWWDB
```

Example 4:

```
INSPECT ITEM TALLYING
  COUNT-0 FOR ALL "AB" AFTER "BA" BEFORE "BC";

INSPECT ITEM REPLACING
  ALL "AB" BY "XY" AFTER "BA" BEFORE "BC"

Initial Value                              Final Value
 of                        COUNT-0           of
ITEM                                       ITEM

ABABABABC                    1             ABABXYABC
```

Example 5:

```
INSPECT ITEM CONVERTING
  "ABCD" TO "XYZX" AFTER QUOTE BEFORE "#".
```

The above INSPECT is equivalent to the following INSPECT:

```
INSPECT item REPLACING
  ALL "A" BY "X" AFTER QUOTE BEFORE "#"
  ALL "B" BY "Y" AFTER QUOTE BEFORE "#"
  ALL "C" BY "Z" AFTER QUOTE BEFORE "#"
  ALL "D" BY "X" AFTER QUOTE BEFORE "#".

Initial Value of ITEM              Final Value of ITEM

AC"AEBDFBCD#AB"D                   AC"XEYXFYZX#AB"D
```

12.5 MERGE

Description

The MERGE statement combines two or more identically sequenced files on a set of specified keys, and during the process makes records available, in merged order, to an output procedure or to an output file.

Format

```
                        {ASCENDING }
MERGE file-name-1 {ON {           } KEY {data-name-1 [FOR DATE]}... }...
                        {DESCENDING}

                              {   alphabet-name }
                              { |------------|  }
                              { | NATIVE     |  }
                              { | STANDARD-1 |  }
    [COLLATING SEQUENCE IS    { | STANDARD-2 |  }]
                              { | ASCII      |  }
                              { | EBCDIC     |  }
                              { | GBCD       |  }
                              { | JIS        |  }
                                |------------|

     USING file-name-2 {file-name-3}...

    {OUTPUT PROCEDURE IS
    {                              {THROUGH}                   }
    {        procedure-name-1 [{          } procedure-name-2]}
    {                              {THRU   }                   }
    {                                                          }
    {GIVING {file-name-4}...                                   }
```

Syntax Rules

1. A MERGE statement may appear anywhere in the Procedure Division except in the declaratives portion or in an input or output procedure associated with a SORT or MERGE statement.

2. File-name-1 must be described in a sort-merge file description entry in the Data Division.

3. If the file referenced by file-name-1 contains variable length records, the size of the records contained in the files referenced by file-name-2 and file-name-3 must not be less than the smallest record nor greater than the largest record described for file-name-1. If the file referenced by file-name-1 contains fixed length records, the size of the records contained in the files referenced by file-name-2 and file-name-3 must not be greater than the largest record described for file-name-1.

4. Data-name-1 is a key data-name. Key data-names are subject to the following rules:

   a. The data items identified by key data-names must be described in records associated with file-name-1.

   b. Key data-names may be qualified.

c. The data items identified by key data-names must not be group items that contain variable occurrence data items.

d. If file-name-1 has more than one record description, the data items identified by key data-names need be described in only one of the record descriptions. The same character positions referenced by a key data-name in one record description entry are taken as the key in all records of the file.

e. None of the data items identified by key data-names can be described by an entry that either contains an OCCURS clause or is subordinate to an entry that contains an OCCURS clause.

f. If the file referenced by file-name-1 contains variable length records, all the data items identified by key data-names must be contained within the first x character positions of the record, where x equals the minimum record size specified for the file referenced by file-name-1.

|g. The data items identified by KEY data-names must not be described as boolean or pointer data items.|

h. If the FOR DATE phrase is specified data-name-1 must be described as PIC 99 USAGE DISPLAY.

5. File-name-2, file-name-3, and file-name-4 must be described in a file description entry, not in a sort-merge description entry, in the Data Division.

6. No two files specified in any one MERGE statement may reside on the same multiple file reel.

7. File-names must not be repeated within the MERGE statement.

8. No pair of file-names in a MERGE statement may be specified in the same SAME AREA or SAME SORT-MERGE AREA clause. The only file-names in a MERGE statement that can be specified in the same SAME RECORD AREA clause are those associated with the GIVING phrase. (See the "SAME AREA Clause".)

9. The words THRU and THROUGH are equivalent.

10. If file-name-4 references an indexed file, the first specification of data-name-1 must be associated with an ASCENDING phrase and the data item referenced by that data-name-1 must occupy the same character positions in its record as the data item associated with the prime record key that file.

11. If the GIVING phrase is specified and the file referenced by file-name-4 contains variable length records, the size of the records contained in the file referenced by file-name-1 must not be less than the smallest record nor greater than the largest record described for file-name-4. If the file referenced by file-name-4 contains fixed length records, the size of the records contained in the file referenced by file-name-1 must not be greater than the largest record described for file-name-4.

12. Procedure-name-1 represents the name of an output procedure.

General Rules

1. The MERGE statement merges all records contained on the files referenced by file-name-2 and file-name-3.

2. If the file referenced by file-name-1 contains only fixed length records, any record in the file referenced by file-name-2 or file-name-3 containing fewer character positions than that fixed length is space filled on the right beginning with the first character position after the last character in the record when that record is released to the file referenced by file-name-1.

3. The data-names following the word KEY are listed from left to right in the MERGE statement in order of decreasing significance without regard to how they are divided into KEY phrases. The leftmost data-name is the major key, the next data-name is the next more significant key, etc.

    a. When the ASCENDING phrase is specified, the merged sequence will be from the lowest value of the contents of the data items identified by the key data-names to the highest value, according to the rules for comparison of operands in a relation condition.

    b. When the DESCENDING phrase is specified, the merged sequence will be from the highest value of the contents of the data items identified by the KEY data-names to the lowest value, according to the rules for comparison of operands in a relation condition.

4. When, according to the rules for the comparison of operands in a relation condition, the contents of all the key data items of one data record are equal to the contents of the corresponding key data items of one or more other data records, the order of return of these records:

    a. follows the order of the associated input files as specified in the MERGE statement.

    b. is such that all records associated with one input file are returned prior to the return of records from another input file.

5. The collating sequence that applies to the comparison of the non-numeric key data items specified is determined at the beginning of the execution of the MERGE statement in the following order of precedence:

    a. First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in that MERGE statement.

    b. Second, the collating sequence established as the program collating sequence.

6. The results of the merge operation are undefined unless the records in the files referenced by file-name-2 and file-name-3 are ordered as described in the ASCENDING or DESCENDING KEY phrases associated with the MERGE statement.

7. All the records in the files referenced by file-name-2 and file-name-3 are transferred to the file referenced by file-name-1. At the start of execution of the MERGE statement, the files referenced by file-name-2 and file-name-3 must not be in the open mode. For each of the files referenced by file-name-2 and file-name-3 the execution of the MERGE statement causes the following actions to be taken:

   a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the INPUT phrase had been executed. If an output procedure is specified, this initiation is performed before control passes to the output procedure.

   b. The logical records are obtained and released to the merge operation. Each record is obtained as if a READ statement with the NEXT and the AT END phrases had been executed.

   c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed. If an output procedure is specified, this termination is not performed until after control passes the last statement in the output procedure.

   These implicit functions are performed such that any associated USE procedures are executed.

8. The output procedure may consist of any procedure needed to select, modify, or copy the records that are made available one at a time by the RETURN statement in merged order from the file referenced by file-name-1. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the output procedure, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the output procedure. The range of the output procedure must not cause the execution of any MERGE, RELEASE, or SORT statement.

9. If an output procedure is specified, control passes to it during execution of the MERGE statement. The compiler inserts a return mechanism at the end of the last statement in the output procedure. When control passes the last statement in the output procedure, the return mechanism provides for termination of the merge, and then passes control to the next executable statement after the MERGE statement. Before entering the output procedure, the merge procedure reaches a point at which it can select the next record in merged order when requested. The RETURN statements in the output procedure are the requests for the next record.

10. During the execution of the output procedure, no statement may be executed manipulating the file referenced by or accessing the record area associated with, file-name-2 or file-name-3. During the execution of any USE AFTER EXCEPTION procedure implicitly invoked while executing the MERGE statement, no statement may be executed manipulating the file referenced by, or accessing the record area associated with, file-name-2, file-name-3, or file-name-4.

11. If the GIVING phrase is specified, all the merged records are written on the file referenced by file-name-4 as the implied output procedure for the MERGE statement. At the start of the execution of the MERGE statement, the file referenced by file-name-4 must not be in the open mode. For each of the files referenced by file-name-4, the execution of the MERGE statement causes the following actions to be taken:

    a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase had been executed.

    b. The merged logical records are returned and written onto the file. Each record is written as if a WRITE statement without any optional phrases had been executed.
    For a relative file, the relative key data item for the first record returned contains the value '1'; for the second record returned, the value '2', etc. After execution of the MERGE statement, the content of the relative key data item indicates the last record returned to the file.

    c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

    These implicit functions are performed such that any associated USE procedures are executed; however, the execution of such a USE procedure must not cause the execution of any statement manipulating the file referenced by, or accessing the record area associated with, file-name-4. On the first attempt to write beyond the externally defined boundaries of the file, any USE AFTER STANDARD EXCEPTION/ERROR procedure specified for the file is executed; if control is returned from that USE procedure or if no such USE procedure is specified, the processing of the file is terminated as in paragraph 11c above.

12. If the file referenced by file-name-4 contains only fixed length records, any record in the file referenced by file-name-1 containing fewer character positions than that fixed length is space filled on the right beginning with the first character position after the last character in the record when that record is returned to the file referenced by file-name-4.

13. Segmentation can be applied to programs containing the MERGE statement. However, the following restrictions apply:

    If the MERGE statement appears in a section that is not in an independent segment, then any output procedure referenced by that MERGE statement must appear totally within non-independent segments, or wholly contained in a single independent segment.

    If a MERGE statement appears in an independent segment, then any output procedure referenced by that MERGE statement must be contained totally within non-independent segments, or wholly within the same independent segment as that MERGE statement.

14. If the FOR DATE phrase is supplied the MERGE proceeds following the "Rule 61" for the corresponding data-name-1 (refer to *SORT/MERGE Utilities User Guide*). The resulting sequence will be:

    ```
    61 62 ... 99 00 01 ... 59 60
    ```

12.6 MOVE

Description

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

Format 1

```
     {identifier-1}
MOVE {            } TO {identifier-2}...
     {literal-1   }
```

Format 2

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

Syntax Rules

1. Literal-1 or the data item referenced by identifier-1 represents the sending area. The data item referenced by identifier-2 represents the receiving area.

2. CORR is an abbreviation for CORRESPONDING.

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

4. An index data item must not appear as an operand of a MOVE statement.

General Rules

1. If the CORRESPONDING phrase is used, selected items within identifier-1 are moved to selected items within identifier-2 according to the rules specified under the appropriate paragraph. (See the CORRESPONDING Phrase.) The results are the same as if the user had referred to each pair of corresponding identifiers in separate format 1 MOVE statements.

2. Literal-1 or the content of the data item referenced by identifier-1 is moved to the data item referenced by each identifier-2 in the order in which it is specified. The rules referring to identifier-2 also apply to the other receiving areas. Any length evaluation or subscripting associated with identifier-2 is evaluated immediately before the data is moved to the respective data item.

   If identifier-1 is reference modified, subscripted, or is a function-identifier, the reference-modifier, subscript, of function-identifier is evaluated only once, immediately before data is moved to the first of the receiving operands. The length of the data item referenced by identifier-1 is evaluated only once, immediately before the data is moved to the first of the receiving operands.

The evaluation of the length of identifier-1 or identifier-2 may be affected by the DEPENDING ON phrase of the OCCURS clause.

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 compiler.

3. Any move in which the receiving operand is an elementary item and the sending operand is either a literal or an elementary item is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, alphanumeric edited, |boolean.| (See the "PICTURE Clause" in Chapter 9.)

   Numeric literals belong to the category numeric, non-numeric literals belong to the category alphanumeric and |boolean literals belong to the category boolean.| The figurative constant ZERO (ZEROS, ZEROES), when moved to a numeric or numeric edited item, belongs to the category numeric. |When moved to a boolean item it belongs to the category boolean.| In all other cases, it belongs to the category alphanumeric. The figurative constant SPACE (SPACES) belongs to the category alphabetic. |The figurative ALL literal, where literal is a boolean literal, belongs to the category boolean.| 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, an alphanumeric edited data item, |a boolean data item,| or an alphabetic data item must not be moved to a numeric or numeric edited data item.

   b. A numeric literal, |a boolean literal,| the figurative constant ZERO, a numeric data item, a numeric edited data item, or |a boolean 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 numeric data item, a numeric edited data item, a numeric literal or figurative constant other than ZERO or ALL literal, where literal is a boolean literal, must not be moved to a boolean data item.

   e. A boolean data item must not be moved to a numeric or numeric edited data item.|

   f. All other elementary moves are legal and are performed according to the rules given in General Rule 4.

4. Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, along with any editing specified for, or de-editing implied by, 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 previously defined. (See "Standard Rules for Data Alignment" in Chapter 3.)

      If the sending operand is described as being signed numeric, the operational sign is not moved; if the operational sign occupies a separate character position, that character is not moved and the size of the sending operand is considered to be one less than its actual size in terms of standard data format characters. (See the "SIGN Clause" in Chapter 9.) If the sending operand is numeric edited, no de-editing takes place. If the usage of the sending operand is different from that of the receiving operand, conversion of the sending operand to the internal representation of the receiving operand takes place. If the sending operand contains the PICTURE symbol 'P', all digit positions specified with this symbol are considered to have the value zero and are counted in the size of the sending operand.

   b. When a numeric or numeric edited item is a receiving item, alignment by decimal point and any necessary zero filling takes place as previously defined except where zeros are replaced because of editing requirements. (See "Standard Rules for Data Alignment" in Chapter 3.) When the sending operand is numeric edited, de-editing is implied to establish the operand's unedited numeric value, which may be signed; then the unedited numeric value is moved to the receiving field.

      (1) When a signed numeric item is the receiving item, the sign of the sending operand is placed in the receiving item. (See the "SIGN Clause" in Chapter 9.) Conversion of the representation of the sign takes place as necessary.

      If the sending operand is unsigned, a positive sign is generated for the receiving item.

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

      (3) When the sending operand is described as being alphanumeric, data is moved as if the sending operand 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 previously defined. (See "Standard Rules for Data Alignment" in Chapter 3.)

   |d. When a boolean item is the receiving item, justification and any necessary boolean character zero filling takes place as previously defined. (See "Standard Rules for Data Alignment" in Chapter 3.)|

5. 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, except as noted in the OCCURS clause. (See the "OCCURS Clause" in Chapter 9.)

6. Data in the following table summarizes the legality of the various types of MOVE statements. The general rule reference indicates the rule that prohibits the move or that describes the behavior of a legal move.

Table 12-1. Legality of Types of MOVE Statements

	CATEGORY OF RECEIVING DATA ITEM			
CATEGORY OF SENDING OPERAND	ALPHABETIC	ALPHANUMERIC EDITED ALPHANUMERIC	BOOLEAN	INTEGER NUMERIC NON-INTEGER NUMERIC EDITED
ALPHABETIC	Yes/4c	Yes/4a	No/3a	No/3a
ALPHANUMERIC	Yes/4c	Yes/4a	Yes/4d	Yes/4b
ALPHANUMERIC EDITED	Yes/4c	Yes/4a	No/3a	No/3a
BOOLEAN	No/3b	Yes/4a	Yes/4d	No/3a
NUMERIC INTEGER	No/3b	Yes/4a	No/3d	Yes/4b
NUMERIC NON-INTEGER	No/3b	No/3c	No/3d	Yes/4b
NUMERIC EDITED	No/3b	Yes/4a	No/3d	Yes/4b

12.7 MULTIPLY

Description

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

Format 1

```
          {identifier-1}
MULTIPLY  {            } BY {identifier-2 [ROUNDED]}...
          {literal-1   }

       [ON SIZE ERROR imperative-statement-1]

       [NOT ON SIZE ERROR imperative-statement-2]

       [END-MULTIPLY]
```

Format 2

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

                GIVING {identifier-3 [ROUNDED]}...

       [ON SIZE ERROR imperative-statement-1]

       [NOT ON SIZE ERROR imperative-statement-2]

       [END-MULTIPLY]
```

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, which is that hypothetical data item resulting from the super-imposition of all receiving data items of a given statement aligned in their decimal points, must not contain more than 18 digits |(up to 30 if the compiler is run with the LEVEL=NSTD parameter).|

General Rules

1. When format 1 is used, literal-1 or the value of the data item referenced by identifier-1 is stored in a temporary data item. The value in this temporary data item is then multiplied by the value of the data item referenced by identifier-2. The value of the multiplier (the value of the data item referenced by identifier-2) is replaced by this product; similarly, the temporary data item is multiplied by each successive occurrence of identifier-2 in the left-to_right order in which identifier-2 is specified.

2. When format 2 is used, literal-1 or the value of the data item referenced by identifier-1 is multiplied by literal-2 or the value of the data item referenced by identifier-2 and the result is stored in the data items referenced by each identifier-3.

3. Additional rules and explanations relative to this statement are given under the appropriate paragraphs. (See "Scope of Statements", the "ROUNDED Phrase", the "SIZE ERROR Phrase", the "Arithmetic Statements", "Overlapping Operands", and "Multiple Results in Arithmetic Statements" in Chapter 10.)

12.8 OPEN

Description

The OPEN statement initiates the processing of files.

Format

```
     {INPUT {file-name-1 [WITH NO REWIND]}... }
OPEN {OUTPUT {file-name-2 [WITH NO REWIND]}...} ...
     {I-O {file-name-3}...                      }
     {EXTEND {file-name-4}...                   }
```

Syntax Rules

1. The OPEN statement for a report file must contain only the OUTPUT phrase or the EXTEND phrase.

2. The NO REWIND phrase must only be used with sequential files. (See the "CLOSE Statement" in Chapter 11.)

3. The I-O phrase must only be used for mass storage files.

4. The EXTEND phrase must only be used for files in the sequential access mode for which neither the MULTIPLE FILE attribute nor the LINAGE clause is specified.

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

General Rules

1. The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode. The successful execution of an OPEN statement associates the file with the file-name through the file connector.

   A file is available if it is physically present and is recognized by the I-O control system. The table below shows the results of opening available and unavailable files.

Table 12-2. Opening Available and Unavailable Files

	FILE IS AVAILABLE	FILE IS UNAVAILABLE
INPUT	Normal OPEN	OPEN is unsuccessful
INPUT (optional file)	Normal OPEN	Normal OPEN; The first READ causes the AT END condition or the INVALID KEY condition.
I-O	Normal OPEN	OPEN is unsuccessful
I-O (optional file)	Normal OPEN	The OPEN causes the file to be created.
OUTPUT	Contents are deleted then normal OPEN	The OPEN causes the file to be created
EXTEND	Normal OPEN	OPEN is unsuccessful
EXTEND (optional file)	Normal OPEN	The OPEN causes the file to be created.

2. The successful execution of an OPEN statement makes the associated record area available to the program. If the file connector associated with file-name is an external file connector, there is only one record area associated with the file connector for the run unit.

3. When a file is not in an open mode, no statement may be executed which references the file, either explicitly or implicitly, except for a MERGE statement with the USING or GIVING phrase, an OPEN statement, or a SORT statement with the USING or GIVING phrases, |or an ASSIGN statement. However, this restriction does not apply if the file has been declared as SYSIN or SYSOUT in the SELECT clause or if it is actually a SYSIN or a SYSOUT file.|

4. The OPEN statement for a report file must be executed prior to the execution of an INITIATE statement for any report contained in the file.

5. An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. In the table "Permissible Access Modes For Different File Organizations" below, 'X' at an intersection indicates that the specified statement, used in the access mode given for that row, may be used with the file organization and open mode given at the top of the column.

6. A file may be opened with the INPUT, OUTPUT, EXTEND and I-O phrases in the same run unit. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement execution for the same file must be preceded by the execution of a CLOSE statement, without the REEL, UNIT or LOCK phrase, for that file.

7. Execution of the OPEN statement does not obtain or release the first data record.

8. If label records are specified for the file, the beginning labels are processed as follow:

    a. When the INPUT phrase is specified, the execution of the OPEN statement causes the labels to be checked.

    b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the labels to be written.

    The behavior of the OPEN statement when label records are specified but not present, or when labels records are not specified but are present, is undefined.

9. If during the execution of an OPEN statement a file attribute conflict condition occurs, the execution of the OPEN statement is unsuccessful.

10. The NO REWIND phrase must only be used with sequential single-reel/unit files. (See the "Close Statement" in Chapter 11.)

11. The NO REWIND phrase will be ignored if it does not apply to the storage media on which the file resides.

12. If the storage medium for the file permits rewinding, the following rules apply:

    a. When neither the EXTEND nor the NO REWIND phrase is specified, execution of the OPEN statement causes the file to be positioned at its beginning.

    b. When the NO REWIND phrase is specified, execution of the OPEN statement does not cause the file to be repositioned; the file must be already positioned at its beginning prior to execution of the OPEN statement.

13. If a file opened with the INPUT phrase is an optional file which is not present, the OPEN statement sets the File Position Indicator to indicate that an optional file is not present.

14. When files are opened with the INPUT or I-O phrase, the file position indicator is set as follow:

    a. For sequential and relative files, the file position indicator is set to 1.

    b. For indexed files, the file position indicator is set to the characters that have the lowest ordinal position in the collating sequence associated with the file and the prime record key is established as the Key of reference.

15. When the EXTEND phrase is specified, the OPEN statement positions the file immediately after the last logical record for that file.

    The last logical record for a file is:

    a. For sequential files, the last record written in the file.

    b. For relative files, the currently existing record with the highest relative record number.

    c. For indexed files, the currently existing record with the highest Prime Key value.

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

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

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

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

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

17. The OPEN statement with the I-O phrase must reference a mass storage file. The execution of the OPEN statement with the I-O phrase places the referenced file in the open mode for both input and output operations.

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

    The labels are checked.

    The new labels are written.

19. For an optional file that is unavailable, the successful execution of an OPEN statement with the EXTEND or I-O phrase creates the file. This creation takes place as if the following statements were executed in the order shown:

    ```
    OPEN OUTPUT file-name. CLOSE file-name.
    ```

    These statements are followed by execution of the OPEN statement specified in the source program.

    The successful execution of an OPEN statement with the OUTPUT phrase creates the file. After the successful creation of a file, that file contains no data records.

20. For sequential files, upon successful execution of the OPEN statement, the Current Volume Pointer is set:

    a. To point to the first or only reel/unit for an available input or input-output file.

    b. To point to the reel/unit containing the last logical record for an extend file.

    c. To point to the new reel/unit for an unavailable output, input-output or extend file.

21. The execution of the OPEN statement causes the value of the I-O status associated with file-name to be updated. (See "I-O Status" in Chapter 7.)

22. If more than one file-name is specified in an OPEN statement, the result of executing this OPEN statement is the same as if a separate OPEN statement had been written for each file-name in the same order as specified in the OPEN statement.

23. The minimum and maximum record sizes for a file are established at the time the file is created and must not subsequently be changed.

Table 12-3. Permissible Access Modes for Different File Organizations

			FILE ORGANIZATION											
			SEQUENTIAL				RELATIVE				INDEXED			
		Open Mode	IN	OUT	I-O	EXT	IN	OUT	I-O	EXT	IN	OUT	I-O	EXT
FILE ACCESS MODE	SEQUENTIAL	READ	X		X		X		X		X		X	
		WRITE		X		X		X		X		X		X
		REWRITE			X				X				X	
		START					X		X		X		X	
		DELETE							X				X	
	RANDOM	READ					X		X		X		X	
		WRITE						X	X			X	X	
		REWRITE							X				X	
		START												
		DELETE							X				X	
	DYNAMIC	READ					X		X		X		X	
		READ NEXT					X		X		X		X	
		WRITE						X	X			X	X	
		REWRITE							X				X	
		START					X		X		X		X	
		DELETE							X				X	

12.9 PERFORM

Description

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. The PERFORM statement is also used to control execution of one or more imperative statements which are within the scope of that PERFORM statement.

Format 1

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

    [imperative-statement-1 END-PERFORM]
```

Format 2

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

    {identifier-1}
    {             } TIMES
    {integer-1    }

    [imperative-statement-1 END-PERFORM]
```

Format 3

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

                {BEFORE}
    [WITH TEST {      }] UNTIL condition-1
                {AFTER }

    [imperative-statement-1 END-PERFORM]
```

Format 4

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

              {BEFORE}
    [WITH TEST {      }]
              {AFTER }

             {identifier-2}      {identifier-3}
     VARYING {             } FROM {index-name-2}
             {index-name-1}      {literal-1   }

                {identifier-4}
              BY {            } UNTIL condition-1
                 {literal-2   }

           {identifier-5}      {identifier-6}
    [AFTER {            } FROM {index-name-4}
           {index-name-3}      {literal-3   }

                {identifier-7}
              BY {            } UNTIL condition-2]...
                 {literal-4   }

    [imperative-statement-1 END-PERFORM]
```

Syntax Rules

1. If procedure-name-1 is omitted, imperative-statement-1 and the END-PERFORM phrase must be specified; if procedure-name-1 is specified, imperative-statement-1 and the END-PERFORM phrase must not be specified.

2. In Format 4, if procedure-name-1 is omitted, the AFTER phrase must not be specified.

3. If neither the TEST BEFORE nor the TEST AFTER phrase is specified, the TEST BEFORE phrase is assumed.

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

5. Each literal represents a numeric literal.

6. The words THRU and THROUGH are equivalent.

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

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

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

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

8. If an index-name is specified in the FROM phrase, then:

   a. The identifier in the associated VARYING or AFTER phrase must reference an integer data item.

   b. The identifier in the associated BY phrase must reference an integer data item.

   c. The literal in the associated BY phrase must be an integer.

9. Literal in the BY phrase must not be zero.

10. Condition-1, condition-2, ... may be any conditional expression (see "Conditional Expressions", Chapter 10).

11. Where procedure-name-1 and procedure-name-2 are both specified and either is the name of a procedure in the declaratives portion of the Procedure Division, both must be procedure-names in the same declarative Section.

General Rules

1. The data items referenced by identifier-4 and identifier-7 must not have a zero value.

2. If an index-name is specified in the VARYING or AFTER phrase, and an identifier is specified in the associated FROM phrase, then the data item referenced by the identifier must have a positive value.

3. When procedure-name-1 is specified, the PERFORM statement is referred to as an out-of-line PERFORM statement; when procedure-name-1 is omitted, the PERFORM statement is referred to as an in-line PERFORM statement.

4. The statements contained within the range of procedure-name-1 (through procedure-name-2 if specified) for an out-of-line PERFORM statement or contained within the PERFORM statement itself for an in-line PERFORM statement are referred to as the specified set of statements.

5. The END-PERFORM phrase delimits the scope of the in-line PERFORM statement. (See "Scope of Statements" in Chapter 10.)

6. An in-line PERFORM statement functions according to the following General Rules for an otherwise identical out-of-line PERFORM statement, with the exception that the statements contained within the in-line PERFORM statement are executed in place of the statements contained within the range of procedure-name-1 (through procedure-name-2 if specified). Unless specially qualified by the word in-line or out-of-line, all the general rules which apply to the out-of-line PERFORM statement also apply to the in-line PERFORM statement.

7. When the PERFORM statement is executed, control is transferred to the first statement of the specified set of statements (except as indicated in General Rules 10b, 10c and 10d). This transfer of control occurs only once for each execution of a PERFORM statement.

For those cases where a transfer of control to the specified set of statements does take place, an implicit transfer of control to the end of the PERFORM statement is established as follows:

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

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

e. If an in-line PERFORM statement is specified, an execution of the PERFORM statement is completed after the last statement contained within it has been executed.

8. 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.

9. If control passes to the specified set of statements by means other than a PERFORM statement, control will pass through the last statement of the set to the next executable statement as if no PERFORM statement referenced the set.

10. The PERFORM statements operate as follows:

    a. Format 1 is the basic PERFORM statement. The specified set of statements referenced by this type of PERFORM statement is executed once, and then control passes to the end of the PERFORM statement.

    b. Format 2 is the PERFORM ... TIMES. The specified set of statements is 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 the 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 end of the PERFORM statement. Following the execution of the specified set of statements the specified number of times, control is transferred to the end of the PERFORM statement.

       During execution of the PERFORM statement, reference to identifier-1 cannot alter the number of times the specified set of statements is to be executed from that which was indicated by the initial value of the data item referenced by identifier-1.

c. Format 3 is the PERFORM ... UNTIL. The specified set of statements is performed until the condition specified by the UNTIL phrase is true. When the condition is true, control is transferred to the end of the PERFORM statement. If the condition is true when the PERFORM statement is entered, and the TEST BEFORE phrase is specified or implied, no transfer to procedure-name-1 takes place, and control is passed to the end of the PERFORM statement. If the TEST AFTER phrase is specified, the PERFORM statement functions as if the TEST BEFORE phrase were specified except that the condition is tested after the specified set of statements has been executed. Any subscripting or reference modification associated with the operands specified in condition-1 is evaluated each time the condition is tested.

d. Format 4 is the PERFORM ... VARYING. This variation of the PERFORM statement is used to augment the values referenced by one or more identifiers or index-names in an orderly fashion during the execution of a PERFORM statement. In the following discussion, every reference to identifier as the object of the VARYING, AFTER, and FROM (current value) phrases also refers to index-names. If index-name-1 or index-name-3 is specified, the value of the associated index at the beginning of the PERFORM statement must be set to an occurrence number of an element in the table. If index-name-2 or index-name-4 is specified, the value of the data item referenced by identifier-2 or identifier-5 at the beginning of the PERFORM statement must be equal to an occurrence number of an element in a table associated with index-name-2 or index-name-4. Subsequent augmentation, as described below, of index-name-1 or index-name-3 must not result in the associated index being set to a value outside the range of the table associated with index-name-1 or index-name-3; except that, at the completion of the PERFORM statement, the index associated with index-name-1 may contain a value that is outside the range of the associated table by one increment or decrement value.

If identifier-2 or identifier-5 is subscripted, the subscripts are evaluated each time the content of the data item referenced by the identifier is set or augmented. If identifier-3, identifier-4, identifier-6, or identifier-7 is subscripted, the subscripts are evaluated each time the content of the data item referenced by the identifier is used in a setting or augmenting operation. Any subscripting or reference modification associated with the operands specified in condition-1 or condition-2 is evaluated each time the condition is tested.

1) If the TEST BEFORE phrase is specified or implied:

When the data item referenced by one identifier is varied, the content of the data item referenced by identifier-2 is set to literal-1 or the current value of the data item referenced by identifier-3 at the point of initial execution of the PERFORM statement; then, if the condition of the UNTIL phrase is false, the specified set of statements is executed once. The value of the data item referenced by identifier-2 is augmented by the specified increment or decrement value (literal-2 or the value of the data item referenced by identifier-4) and condition-1 is evaluated again. The cycle continues until this condition is true, at which point control is transferred to the end of the PERFORM statement. If condition-1 is true at the beginning of execution of the PERFORM statement, control is transferred to the end of the PERFORM statement.

When the data items referenced by two identifiers are varied, the content of the data item referenced by identifier-2 is set to literal-1 or the current value of the data item referenced by identifier-3 and then the content of the data item referenced by identifier-5 is set to literal-3 or the current value of the data item referenced by identifier-6. After the contents of the data items referenced by the identifiers have been set, condition-1 is evaluated; if true, control is transferred to the end of the PERFORM statement; if false, condition-2 is evaluated. If condition-2 is false, the specified set of statements is executed once, then the content of the data item referenced by identifier-5 is augmented by literal-4 or the content of the data item referenced by identifier-7 and condition-2 is evaluated again. This cycle of evaluation and augmentation continues until this condition is true. When condition-2 is true, the content of the data item referenced by identifier-2 is augmented by literal-2 or the content of the data item referenced by identifier-4, the content of the data item referenced by identifier-5 is set to literal-3 or the current value of the data item referenced by identifier-6, and condition-1 is reevaluated. The PERFORM statement is completed if condition-1 is true; if not, the cycle continues until condition-1 is true.

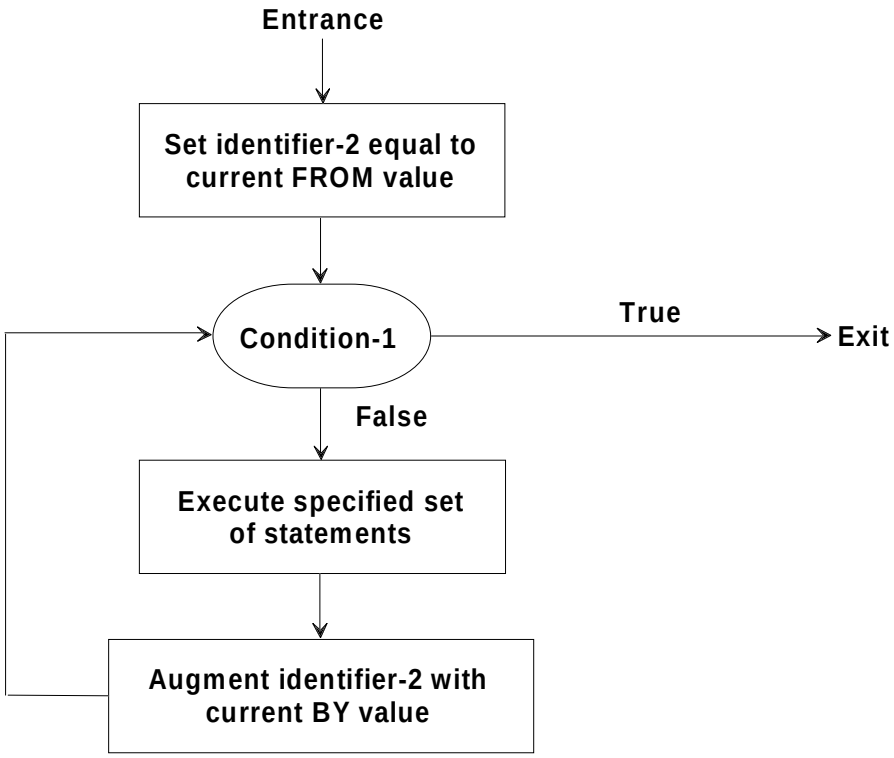


Figure 12-1. Perform Test before Varying with One Condition

At the termination of the PERFORM statement, the data item referenced by identifier-5 contains literal-3 or the current value of the data item referenced by identifier-6. The data item referenced by identifier-2 contains a value that exceeds the last used setting by one increment or decrement value, unless condition-1 was true when the PERFORM statement was entered, in which case, the data item referenced by identifier-2 contains literal-1 or the current value of the data item referenced by identifier-3.

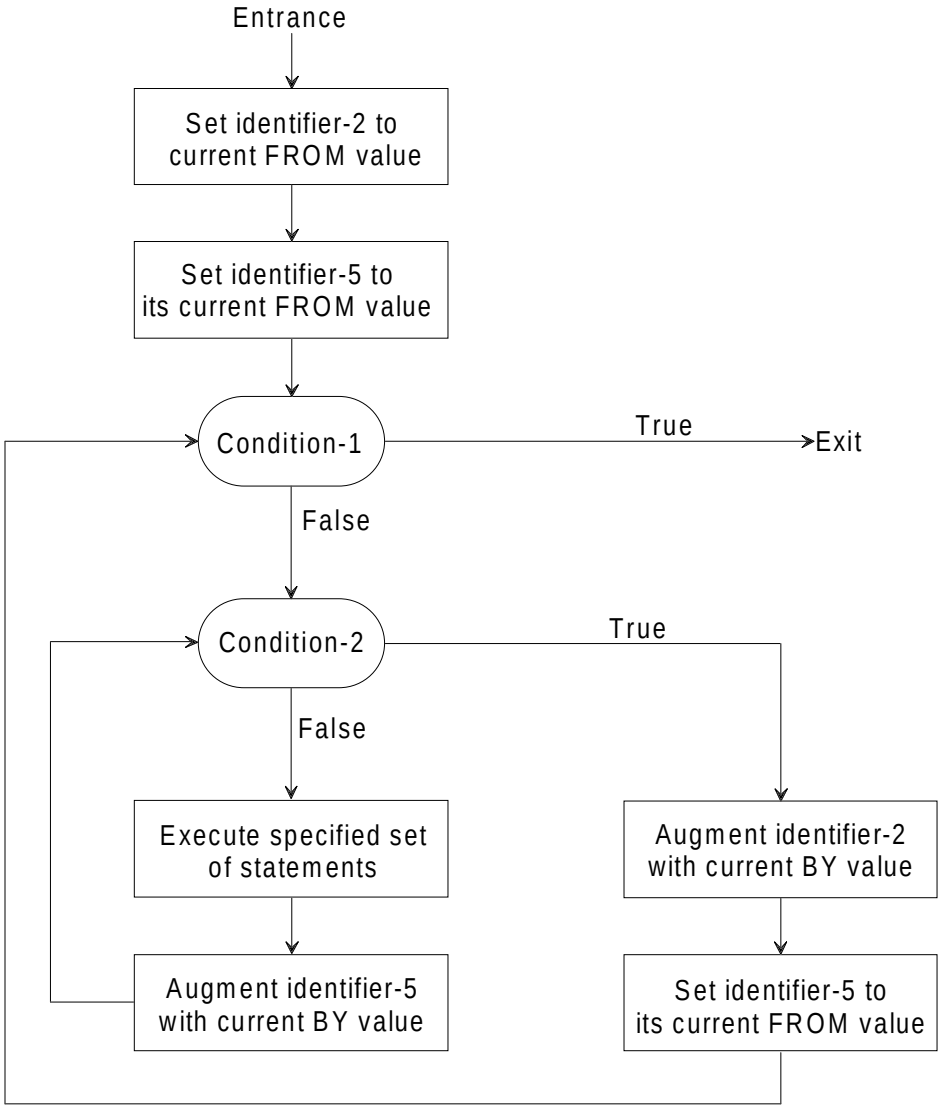


Figure 12-2. Perform Test before Varying with Two Conditions

2) If the TEST AFTER phrase is specified:

When the data item referenced by one identifier is varied, the content of the data item referenced by identifier-2 is set to literal-1 or the current value of the data item referenced by identifier-3 at the point of execution of the PERFORM statement; then the specified set of statements is executed once and condition-1 of the UNTIL phrase is tested. If the condition is false, the value of the data item referenced by identifier-2 is augmented by the specified increment or decrement value (literal-2 or the value of the data item referenced by identifier-4) and the specified set of statements is executed again. The cycle continues until condition-1 is tested and found to be true, at which point control is transferred to the end of the PERFORM statement.

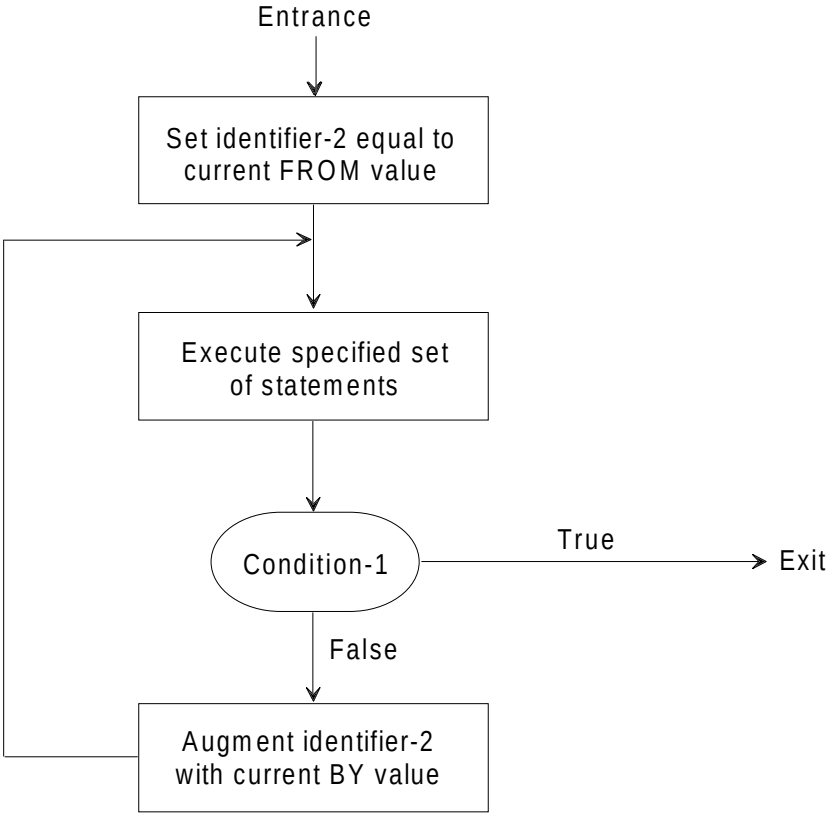


Figure 12-3. Perform Test after Varying with One Condition

When the data items referenced by two identifiers are varied, the content of the data item referenced by identifier-2 is set to literal-1 or the current value of the data item referenced by identifier-3, then the content of the data item referenced by identifier-5 is set to literal-3 or the current value of the data item referenced by identifier-6, and the specified set of statements is then executed. Condition-2 is then evaluated; if false, the content of the data item referenced by identifier-5 is augmented by literal-4 or the content of the data item referenced by identifier-7 and the specified set of statements is again executed. The cycle continues until condition-2 is again evaluated and found to be true, at which time condition-1 is evaluated. If false, the content of the data item referenced by identifier-2 is augmented by literal-2 or the content of the data item referenced by identifier-4, the content of the data item referenced by identifier-5 is set to literal-3 or the current value of the data item referenced by identifier-6 and the specified set of statements is again executed. This cycle continues until condition-1 is again evaluated and found to be true, at which time control is transferred to the end of the PERFORM statement.

After completion of the PERFORM statement, each data item varied by an AFTER or VARYING phrase contains the same value it contained at the end of the most recent execution of the specified set of statements.

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

When the data items referenced by two identifiers are varied, the data item referenced by identifier-5 goes through a complete cycle (FROM, BY, UNTIL) each time the content of the data item referenced by identifier-2 is varied. When the contents of three or more data items referenced by identifiers are varied, the mechanism is the same as for two identifiers except that the data item being varied by each AFTER phrase goes through a complete cycle each time the data item being varied by the preceding AFTER phrase is augmented.

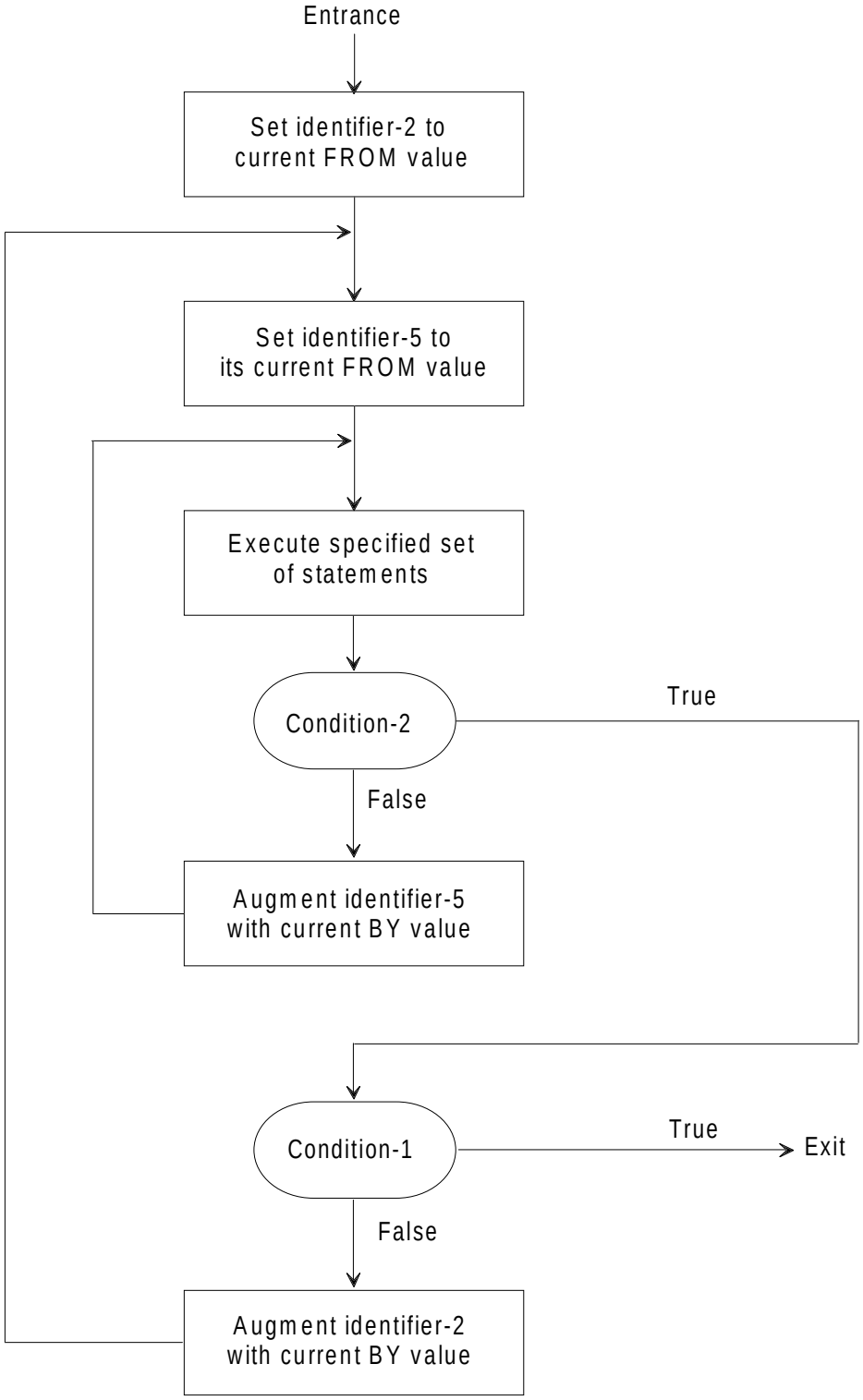


Figure 12-4. Perform Test after Varying with Two Conditions

11. The range of a PERFORM statement consists logically of all those statements that are executed as a result of executing the PERFORM statement through execution of the implicit transfer of control to the end of the PERFORM statement. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the PERFORM statement, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the PERFORM statement. The statements in the range of a PERFORM statement need not appear consecutively in the source program.

12. Statements executed as the result of a transfer of control caused by executing an EXIT PROGRAM statement are not considered to be part of the range of the PERFORM statement when:

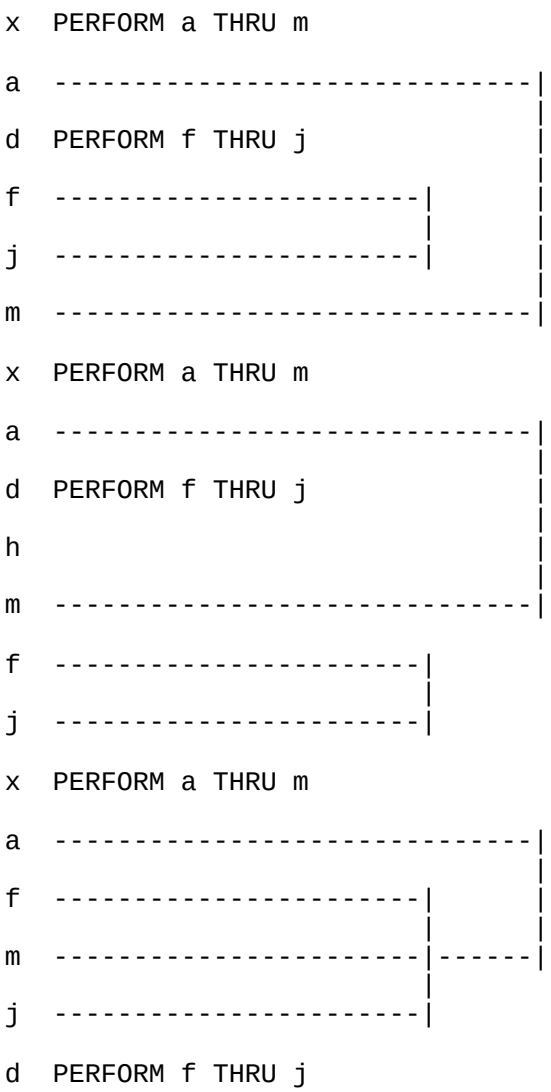
    a. That EXIT PROGRAM statement is specified in the same program in which the PERFORM statement is specified, and

    b. The EXIT PROGRAM statement is within the range of the PERFORM statement.

13. Procedure-name-1 and procedure-name-2 must not name sections or paragraphs in any other program in the run unit, irrespective of whether or not the other program contains or is contained within the program which includes the PERFORM statement. Statements in other programs in the run unit may only be obeyed as a result of executing a PERFORM statement, if the range of that PERFORM statement includes CALL and EXIT PROGRAM statements.

14. If the range of 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 the following illustrations for examples of legal PERFORM constructs:



15. 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:

    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.

16. 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.

12.10 PURGE

Description

The PURGE statement eliminates from the message control system (MCS) a partial message which has been released by one or more SEND statements.

Format

```
PURGE cd-name-1
```

Syntax Rules

Cd-name-1 must reference an output CD or input-output CD.

General Rules

1. Execution of a PURGE statement causes the MCS to eliminate any partial message awaiting transmission to the destinations specified in the CD referenced by cd-name-1.

2. Any message that has associated with it an EMI or EGI is not affected by the execution of a PURGE statement.

3. The content of the status key data item and the content of the error key data item (if applicable) of the area referenced by cd-name-1 are updated by the MCS.

12.11 READ

Description

1. For sequential access, the READ statement makes available the next |or previous| logical record from a file.

2. For random access, the READ statement makes available a specified record from a mass storage file.

Format 1

```
                      |-------------|
                    [{|  PREVIOUS  }|]
                      |-------------|
READ file-name-1    [{             } ] RECORD [INTO identifier-1]
                    [{     NEXT    } ]

     [AT END imperative-statement-1]

     [NOT AT END imperative-statement-2]

     [END-READ]
```

Format 2

```
READ file-name-1 RECORD [INTO identifier-1] [KEY IS data-name-1]

[INVALID KEY imperative-statement-1]

[NOT INVALID KEY imperative-statement-2]

[END-READ]
```

Syntax Rules

1. The storage area associated with identifier-1 and the record area associated with file-name-1 must not be the same storage area.

2. Data-name-1 must be the name of a data item specified as a Record Key associated with file-name-1.

3. Data-name-1 may be qualified.

4. Format 1 must be used for all files in sequential access mode.

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

6. Format 2 is used for files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.

7. The INVALID KEY or the AT END phrase must be specified if no applicable USE AFTER STANDARD EXCEPTION procedure is specified for file-name-1.

8. File-name-1 must not be the name of a report file or the name of a sort or merge file.

|9. The PREVIOUS phrase applies only to indexed files.|

General Rules

1. The file referenced by file-name-1 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. For files in the sequential access mode, the NEXT phrase is optional and has no effect on the execution of the READ statement.

3. The execution of the READ statement causes the value of the I-O status associated with file-name-1 to be updated.

4. The setting of the file position indicator at the start of the execution of a format-1 READ statement is used in determining the record to be made available according to the following rules.
   Comparisons for records in sequential files relate to the record number. Comparisons for records in relative files relate to the relative key number. Comparisons for records in indexed files relate to the value of the current key of reference. For indexed files, the comparisons are made according to the collating sequence of the file.

   a. If the file position indicator indicates that no valid next record has been established, execution of the READ statement is unsuccessful.

   b. If the file position indicator indicates that an optional input file is not present or that the at end condition already exists, execution proceeds as specified in general rule 11.

   c. If the file position indicator indicates that no next logical record exists, the file position indicator is set to indicate that the at end condition already exists and execution proceeds as specified in general rule 11.

   d. If the file position indicator was established by a previous OPEN or Format 1 START statement, the first existing record in the file whose record number, relative record number, or key value is greater than or equal to the file position indicator is selected.

      |If the file position indicator was established by a previous Format 2 START statement, it is the last existing record in the file whose key value is equal to the file position indicator which is selected.|

   e. If the file position indicator was established by a previous READ statement, and the current key of reference (if any) does not allow duplicates. |When the PREVIOUS phrase is used, the last existing record in the file whose key value is less than the file position indicator is selected, otherwise| the first existing record in the file whose record number, relative record number, or key value is greater than the file position indicator is selected.

f. For relative files, if the RELATIVE KEY phrase is specified for file-name-1 and the number of significant digits in the relative record number of the selected record is larger than the size of the relative key data item, the result will be unpredictable.

g. For indexed files, if the file position indicator was established by a previous READ statement and the current key of referenced does allow duplicates. |When the PREVIOUS phrase is used, the last record in the file whose key value is either equal to the file position indicator and whose logical position within the set of duplicates is immediately before the record that was made available by that previous READ statement or whose key value is less than the file position indicator is selected, otherwise| the first record in the file whose key value is either equal to the file position indicator and whose logical position within the set of duplicates is immediately after the record that was made available by the previous READ statement or whose key value is greater than the file position indicator, is selected.

If a record is found which satisfies the above rules, it is made available in the record area associated with file-name-1.

If no record is found which satisfies the above rules, the file position indicator is set to indicate that no next logical record exists and execution proceeds as specified in general rule 11.

If a record is made available, the file position indicator is set to the record number or the value of the current key of reference of the record made available.

5. Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged; a record is available to the object program prior to the execution of imperative-statement-2, if specified, or prior to the execution of any statement following the READ statement, if imperative-statement-2 is not specified.

6. When the logical records of a file are described with more than one record description, these records automatically share the same record area in storage; this is equivalent to an implicit re-definition 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.

7. The INTO phrase may be specified in a READ statement:

   a. If only one record description is subordinate to the file description entry, or

   b. If all record-names associated with file-name-1 and the data item referenced by identifier-1 describe a group item or an elementary alphanumeric item.

8. The result of the execution of a READ statement with the INTO phrase is equivalent to the application of the following rules in the order specified:

   a. The execution of the same READ statement without the INTO phrase.

   b. The current record is moved from the record area to the area specified by identifier-1 according to the rules for the MOVE statement without the CORRESPONDING phrase. The size of the current record is determined by the rules specified for the RECORD clause. If the file description entry contains a RECORD VARYING clause, the implied move is a group move. The implied MOVE statement does not occur if the execution of the READ statement was unsuccessful. Any subscripting associated with identifier-1 is evaluated after the record has been read and immediately before it is moved to the data item. The record is available in both the record area and the data item referenced by identifier-1.

9. For sequential READ, if, during the execution of a READ statement, the end of reel/unit is recognized or a reel/unit contains no logical records, and the logical end of the file has not been reached, the following operations are executed:

   a. The standard ending reel/unit label procedure.

   b. A reel/unit swap. The current volume pointer is updated to point to the next reel/unit existing for the file.

   c. The standard beginning reel/unit label procedure.

10. If at the time of the execution of a format 2 READ statement, the file position indicator indicates that an optional input file is not present, the invalid key condition exists and execution of the READ statement is unsuccessful.

11. For a format 1 READ statement, if the file position indicator indicates that no next logical record exists, or that an optional input file is not present, or that the at end condition already exists, the following occurs in the order specified:

    a. A value, derived from the setting of the file position indicator, is placed into the I-O status associated with file-name-1 to indicate the at end condition.

    b. If the AT END phrase is specified in the statement causing the condition, control is transferred to imperative-statement-1 in the AT END phrase. Any USE AFTER STANDARD EXCEPTION procedure associated with file-name-1 is not executed.

    c. If the AT END phrase is not specified, a USE AFTER STANDARD EXCEPTION procedure must be associated with file-name-1, and that procedure is executed. Return from that procedure is to the next executable statement following the end of the READ statement.

    When the at end condition occurs, execution of the READ statement is unsuccessful.

12. If neither an at end nor an invalid key condition occurs during the execution of a READ statement, the AT END phrase or the INVALID KEY phrase is ignored, if specified, and the following actions occur:

    a. The file position indicator is set and the I-O status associated with file-name-1 is updated.

    b. If an exception which is not an at end or an invalid key condition exists, control is transferred according to the rules of the USE statement following the execution of any USE AFTER EXCEPTION procedure applicable to file-name-1.

    c. If no exception condition exists, the record is made available in the record area and any implicit move resulting from the presence of an INTO phrase is executed. Control is transferred to the end of the READ statement or to imperative-statement-2, if specified. In the latter case, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the READ statement.

13. Following the unsuccessful execution of a READ statement, the content of the associated record area is undefined, the key of reference is undefined for indexed files, and the file position indicator is set as follow:

    a. For a format 2 READ statement, the file position indicator is set to indicate that no valid next record has been established.

    b. For a format 1 READ statement, the following rules apply:

       If the file position indicator indicates that no next logical record exists or that the at end condition already exists, the file position indicator is unchanged.

       If the file position indicator indicates that an optional input file is not present, the file position indicator is set to indicate that the at end condition already exists.

       In all other cases, the file position indicator is set to indicate that no valid next record has been established.

14. For a relative file if the RELATIVE KEY phrase is specified for file-name-1, the execution of a format 1 READ statement moves the relative record number of the record made available to the relative key data item according to the rules for the MOVE statement.

15. For a relative file, the execution of a Format 2 READ statement sets the file position indicator to the value contained in the data item referenced by the RELATIVE KEY phrase for the file, and the record whose relative record number equals the file position indicator is made available in the record area associated with file-name-1. If the file does not contain such a record, the invalid key condition exists and execution of the READ statement is unsuccessful. (See the "Invalid Key Condition", Chapter 10.)

16. For a relative or indexed 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 that file.

    |For an indexed file for which dynamic access mode is specified, a Format 1 READ statement with the PREVIOUS phrase specified causes the previous logical record to be retrieved from that file.|

17. For an indexed file being sequentially accessed |and when the PREVIOUS phrase is not used|, records having the same duplicate value in an Alternate Record Key which is the Key of Reference are made available in the same order in which they are released by execution of WRITE statements, or by execution of REWRITE statements which create such duplicate values.

    |When the PREVIOUS phrase is used, they are made available in the reverse order.|

18. For an indexed file if the KEY phrase is specified in a Format 2 READ statement, data-name-1 is established as the Key of Reference for this retrieval. If the dynamic access mode is specified, this Key of Reference is also used for retrievals by any subsequent executions of Format 1 READ statements for the file until a different Key of Reference is established for the file.

19. For an indexed file, if the KEY phrase is not specified in a Format 2 READ statement, the Prime Record Key is established as the key of reference for this statement. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent execution of Format 1 READ statements for the file until a different key of reference is established for the file.

20. For an indexed file, execution of a Format 2 READ statement sets the file position indicator to the value in the key of reference. This value is compared with the value contained in the corresponding data item of the stored records in the file, until the first record having an equal value is found. In the case of an alternate key with duplicate values, the first record found is the first record of a sequence of duplicates which was released to the Operating System. The record so found is made available in the record area associated with file-name-1. If no record can be so identified, the invalid key condition exists and execution of the READ statement is unsuccessful. (See the "Invalid Key Condition", Chapter 10.)

21. If the number of character positions in the record that is read is less than the minimum size specified by the record description entries for file-name-1, the portion of the record area which is to the right of the last valid character read is undefined. If the number of character positions in the record that is read is greater than the maximum size specified by the record description entries for file-name-1, the record is truncated on the right to the maximum size. In either of these cases, the READ statement is successful and the I-O status is set indicating a record length conflict has occurred.

22. The END-READ phrase delimits the scope of the READ statement.A description of the function of the END-READ phrase is given in the appropriate paragraph. (See "Scope of Statements" in Chapter 10.)

12.12 RECEIVE

Description

The RECEIVE statement makes available a message or a message segment and information about that data.

The RECEIVE statement allows a specific imperative statement when no data is available.

Format

```
                    {MESSAGE}
RECEIVE cd-name-1 {         } INTO identifier-1
                    {SEGMENT}

      [NO DATA imperative-statement-1]

      [WITH DATA imperative-statement-2]

      [END-RECEIVE]
```

Syntax Rules

1. Cd-name-1 must reference an input CD or input-output CD.

General Rules

1. If cd-name-1 references an input CD, the contents of the data items specified by data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name-1 designate the queue structure containing the message. (See the "Communication Description - Complete Entry Skeleton" in Chapter 8.)

2. If cd-name-1 references an input-output CD, the content of the data item specified by data-name-3 (SYMBOLIC TERMINAL) of the area referenced by cd-name-1 designates the source of the message. (See the "Communication Description - Complete Entry Skeleton" in Chapter 8.)

3. The message, message segment, or portion of a message or segment is transferred to the receiving character positions of the area referenced by identifier-1 aligned to the left without space fill.

4. When during the execution of a RECEIVE statement, the MCS makes data available in the data item referenced by identifier-1, the NO DATA phrase, if specified, is ignored and control is transferred to the end of the RECEIVE statement or, if the WITH DATA phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the RECEIVE statement.

5. When, during the execution of a RECEIVE statement, the MCS does not make data available in the data item referenced by identifier-1, one of the three actions listed below will occur.

   a. If the NO DATA phrase is specified in the RECEIVE statement, the RECEIVE operation is terminated with the indication that action is complete and control is transferred to imperative-statement-1. Execution then continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the RECEIVE statement and the WITH DATA phrase, if specified, is ignored.

   b. If the NO DATA phrase is not specified in the RECEIVE statement, execution of the object program is suspended until data is made available in the data item referenced by identifier-1.

   c. If one or more queues or subqueues are unknown to the MCS, the appropriate status key code is stored and control is then transferred as if data had been made available. (See "Communication Status Key Condition", Chapter 8).

6. The data items identified by cd-name-1 are appropriately updated by the Message Control System (MCS) at each execution of a RECEIVE statement (See the "Communication Description", Chapter 8).

7. A single execution of a RECEIVE statement never returns to the data item referenced by identifier-1 more than a single message (when the MESSAGE phrase is used) or a single segment (when the SEGMENT phrase is used). However, the MCS does not pass any portion of a message to the object program until the entire message is available to the MCS, even if the SEGMENT phrase of the RECEIVE statement is specified.

8. When the MESSAGE phrase is used, end of segment indicators are ignored, and the following rules apply to the data transfer:

   a. If a message is the same size as the area referenced by identifier-1, the message is stored in the area referenced by identifier-1.

   b. If a message size is less than the area referenced by identifier-1, the message is aligned to the leftmost character position of the area referenced by identifier-1 and the contents of the character positions not occupied by characters of the message are not changed.

   c. If a message size is greater than the area referenced by identifier-1 the message fills the area referenced by identifier-1 left to right starting with the leftmost character of the message. Further RECEIVE statements which reference the same queue and sub-queue must be executed to transfer the remainder of the message into the area referenced by identifier-1. The remainder of the message, for the purposes of applying rules 8a, 8b and 8c is treated as a new message.

   d. If an end of group indicator is associated with the text accessed by the RECEIVE statement, the existence of an end of message indicator is implied.

9. When the SEGMENT phrase is used, the following rules apply:

   If a segment is the same size as the area referenced by identifier-1, the segment is stored in the area referenced by identifier-1.

   If the segment size is less than the area referenced by identifier-1, the segment is aligned to the leftmost character position of the area referenced by identifier-1 and the contents of character positions not occupied by characters of the segment are not changed.

   If a segment size is greater than the area referenced by identifier-1, the segment fills the area referenced by identifier-1 left to right starting with the leftmost character of the segment. Further RECEIVE statements which reference the same queue, sub-queue, ... , must be executed to transfer the remainder of the segment into the area referenced by identifier-1. The remainder of the segment, for the purpose of applying rules 9a, 9b, and 9c, is treated as a new segment.

   If an end of message indicator or end of group indicator is associated with the text accessed by the RECEIVE statement, the existence of an end of segment indicator is implied.

10. Once the execution of a RECEIVE statement has returned a portion of a message, only subsequent execution of RECEIVE statements in that run unit can cause the remaining portion of the message to be returned.

11. The END-RECEIVE phrase delimits the scope of the RECEIVE statement.

12.13 RELEASE

Description

The RELEASE statement transfers records to the initial phase of a SORT operation.

Format

```
RELEASE record-name-1 [FROM identifier-1]
```

Syntax Rules

1. Record-name-1 must be the name of a logical record in a Sort-Merge File Description entry and it may be qualified.

2. A RELEASE statement may be used only within the range of an input procedure associated with a SORT statement for the file-name whose sort-merge file description entry contains record-name-1.

3. If identifier-1 is a function-identifier, it must reference an alphanumeric function. When identifier-1 is not a function-identifier, record-name-1 and identifier-1 must not refer to the same storage area.

General Rules

1. The execution of a RELEASE statement causes the record named by record-name-1 to be released to the initial phase of a sort operation.

2. The logical record released by the execution of the RELEASE statement is no longer available in the record area unless the sort-merge file-name associated with record-name-1 is specified in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as the file associated with record-name-1.

3. The result of the execution of a RELEASE statement with the FROM phrase is equivalent to the execution of the following statements in the order specified:

   a. The statement: `MOVE identifier-1 to record-name-1` according to the rules specified for the MOVE statement.

   b. The same RELEASE statement without the FROM phrase.

4. After the execution of the RELEASE 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-1 is not available except as specified by the SAME RECORD AREA clause.

12.14 RETURN

Description

The RETURN statement obtains either sorted records from the final phase of a SORT operation or merged records during a MERGE operation.

Format

```
RETURN file-name-1 RECORD [INTO identifier-1]
        AT END imperative-statement-1
       [NOT AT END imperative-statement-2]
       [END-RETURN]
```

Syntax Rules

1. The storage area associated with identifier-1 and the record area associated with file-name-1 must not be the same storage area.

2. File-name-1 must be described by a sort-merge file description entry in the Data Division.

3. A RETURN statement may only be used within the range of an output procedure associated with a SORT or MERGE statement for file-name-1.

General Rules

1. When the logical records in a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit re-definition 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 RETURN statement.

2. The execution of the RETURN statement causes the next existing record in the file referenced by file-name-1, as determined by the keys listed in the SORT or MERGE statement, to be made available in the record area associated with file-name-1. If no next logical record exists in the file referenced by file-name-1, the at end condition exists and control is transferred to imperative-statement-1 of the AT END phrase. Execution continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred according to the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the RETURN statement and the NOT AT END phrase is ignored, if specified. When the at end condition occurs, execution of the RETURN statement is unsuccessful and the contents of the record area associated with file-name-1 are undefined. After the execution of imperative-statement-1 in the AT END phrase, no RETURN statement may be executed as part of the current output procedure.

3. If an at end condition does not occur during the execution of a RETURN statement, then after the record is made available and after executing any implicit move resulting from the presence of an INTO phrase, control is transferred to imperative-statement-2, if specified; otherwise, control is transferred to the end of the RETURN statement.

4. The END-RETURN phrase delimits the scope of the RETURN statement.

5. The INTO phrase may be specified in a RETURN statement if only one record description is subordinate to the sort-merge file description entry, or if all record-names associated with file-name-1 and the data item referenced by identifier-1 describe a group item or an elementary alphanumeric item.

6. The result of the execution of a RETURN statement with the INTO phrase is equivalent to the application of the following rules in the order specified:

    a. The execution of the same RETURN statement without the INTO phrase.

    b. The current record is moved from the record area to the area specified by identifier-1 according to the rules for the MOVE statement without the CORRESPONDING phrase. The size of the current record is determined by rules specified for the RECORD clause. If the file description entry contains a RECORD VARYING clause, the implied move is a group move. The implied MOVE statement does not occur if the execution of the RETURN statement was unsuccessful. Any subscripting associated with identifier-1 is evaluated after the record has been read and immediately before it is moved to the data item. The record is available in both the record area and the data item referenced by identifier-1.

12.15 REWRITE

Description

The REWRITE statement logically replaces a record existing in a mass storage file.

Format

```
REWRITE record-name-1 [FROM identifier-1]
        [INVALID KEY imperative-statement-1]
        [NOT INVALID KEY imperative-statement-2]
        [END-REWRITE]
```

Syntax Rules

1. If identifier-1 is a function-identifier, it must reference an alphanumeric function. When identifier-1 is not a function-identifier, record-name-1 and identifier-1 must not refer to the same storage area.

2. Record-name-1 is the name of a logical record in the File Section of the Data Division and may be qualified.

3. The INVALID KEY and the NOT INVALID KEY phrases must not be specified for a REWRITE statement which references a sequential file or a relative file in sequential access mode.

4. The INVALID KEY phrase must be specified in REWRITE statement for relative and indexed files in the random or dynamic access mode, and for which an appropriate USE AFTER STANDARD EXCEPTION procedure is not specified.

5. Record-name-1 must not be defined within a Sort or Sort-Merge File Description entry.

General Rules

All Files

1. The file referenced by the file-name associated with record-name-1 must be a mass storage file and must be open in the I-O mode at the time of execution of this statement (see the "OPEN Statement" in Chapter 12).

Sequential Files

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-1 may or may not 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 unless the file-name associated with record-name-1 is specified in a SAME RECORD AREA clause. The logical record is available to the program as a record of other files referenced in the SAME RECORD AREA clause as the associated output file, as well as the file associated with record-name-1.

5. The result of the execution of a REWRITE statement with the FROM phrase is equivalent to the execution of the following statements in the order specified:

   a. The statement: MOVE identifier-1 to record-name-1 according to the rules specified for the MOVE statement.

   b. The same REWRITE statement without the FROM phrase.

6. After the execution of the REWRITE 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-1 is not available except as specified by the SAME RECORD AREA clause.

7. The file position indicator is not affected by the execution of a REWRITE statement.

8. The execution of the REWRITE statement causes the value of the I-O status of the file-name associated with record-name-1, to be updated. (see "I-O Status" in Chapter 7).

9. The execution of the REWRITE statement releases a logical record to the operating system.

10. Transfer of control following the successful or unsuccessful execution of the REWRITE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the REWRITE statement. (See the "Invalid Key Condition" in Chapter 10.)

11. The END-REWRITE phrase delimits the scope of the REWRITE statement. A description of the function of the END-REWRITE phrase is given in the appropriate paragraph. (See "Scope of Statements" in Chapter 10.)

12. For sequential files, the number of character positions in the record referenced by record-name-1 must not be unequal to the number of character positions in the record being replaced.

    For relative and indexed files, the number of character positions in the record referenced by record-name-1 must not be larger than the largest or smaller than the smallest number of character positions allowed by the RECORD IS VARYING clause associated with the file-name associated with record-name-1.

    In either of these cases the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the content of the record area is unaffected and the I-O status of the file associated with record-name-1 is set to a value indicating the cause of the condition.

Relative Files

13. For a file accessed in either random or dynamic access mode, the Operating System logically replaces the record specified by the content of the RELATIVE KEY data item of the file-name associated with record-name-1. If the file does not contain the record specified by the key, the invalid key condition exists. When the invalid key condition is recognized, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the contents of the record area are unaffected and the I-O status of the file-name associated with record-name-1 is set to a value indicating the cause of the condition. (See "I-O Status" in Chapter 7, the "Invalid Key Condition" in Chapter 10.)

Indexed Files

14. For a file in the sequential access mode, the record to be replaced by a REWRITE statement is specified by the value of the Prime Record Key. When the REWRITE statement is executed the value of the Prime Record Key of the record to be replaced must be equal to the value of the Prime Record Key of the last record read from this file.

15. For a file in the random or dynamic access mode, the record to be replaced by the REWRITE statement is specified by the Prime Record Key.

16. Execution of the REWRITE statement for a record which has an Alternate Record Key occurs as follows:

    a. When the value of a specific Alternate Record Key is not changed, the order of retrieval when that key is the Key of Reference remains unchanged.

    b. When the value of a specific Alternate Record Key is changed, the subsequent order of retrieval of that record may be changed when that specific Alternate Record Key is the Key of Reference. When duplicate key values are permitted, the record is logically positioned last within the set of duplicate records containing the same Alternate Record Key value as the Alternate Record Key value that was placed in the record.

17. The invalid key condition exists under the following circumstances:

    a. When the file is open in the sequential access mode, and the value of the Prime Record Key of the record to be replaced is not equal to the value of the Prime Record Key of the last record read from the file, or

    b. When the file is open in the dynamic or random access mode, and the value of the Prime Record Key of the record to be replaced is not equal to the value of the Prime Record Key of any record existing in the file, or

    c. When the value of an ALTERNATE RECORD KEY of the record to be replaced, for which duplicates are not allowed, equals the value of the corresponding data item of a record already existing in the file.

18. When the invalid key condition is recognized, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the content of the record area is unaffected and the I-O status of the file-name associated with record-name-1 is set to a value indicating the cause of the condition.

13. Procedure Division - Statements (SEARCH to WRITE)

This chapter describes the statements from SEARCH to WRITE (inclusive).

The statements concerned are:

- SEARCH
- SEND
- SET
- SORT
- START
- STOP
- STRING
- SUBTRACT
- SUPPRESS
- TERMINATE
- |TRANSFORM|
- UNSTRING
- USE
- WRITE

13.1 SEARCH

Description

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

Format 1

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

    [AT END imperative-statement-1]

                      {imperative-statement-2}
    {WHEN CONDITION-1 {                      }}... [END-SEARCH]
                      {NEXT SENTENCE         }
```

Format 2

```
SEARCH ALL identifier-1

   [AT END imperative-statement-1]

         {            {IS EQUAL TO}
         {            {|--------| } {identifier-3           }}
         {data-name-1 {| EQUALS | } {literal-1              }}
    WHEN {            {|--------| } {arithmetic-expression-1}}
         {            {IS =      }
         {                                                  }
         {condition-name-1                                  }

         {            {IS EQUAL TO}
         {            {|--------| } {identifier-4           }}
         {data-name-2 {| EQUALS | } {literal-2              }}
   [AND  {            {|--------| } {arithmetic-expression-2}}]...
         {            {IS =      }
         {                                                  }
         {condition-name-2                                  }

   {imperative-statement-2}
   {                     } [END-SEARCH]
   {NEXT SENTENCE        }
```

Syntax Rules

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

2. Identifier-2, when specified, must reference a data item described as USAGE IS INDEX or as a numeric elementary data item without any positions to the right of the assumed decimal point. Identifier-2 may not be subscripted by the first (or only) index-name specified in the INDEXED BY phrase in the OCCURS clause associated with identifier-1.

3. In Format 1, condition-1 may be any conditional expression (see "Conditional Expressions", Chapter 10).

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

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

5. If the END-SEARCH phrase is specified, the NEXT SENTENCE phrase must not be specified.

6. The words IS EQUAL TO are equivalent to the words IS = |, and both are equivalent to the word EQUALS.|

General Rules

1. The scope of a SEARCH statement may be terminated by any of the following:

   a. An END-SEARCH phrase at the same level of nesting.

   b. A separator period.

   c. An ELSE or END-IF phrase associated with a previous IF statement. (See "Scope of Statements" in Chapter 10.)

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

   a. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is greater than the highest permissible occurrence number for identifier-1, the SEARCH is terminated immediately. The number of occurrences of identifier-1, the last of which is the highest permissible, is discussed in the OCCURS clause. (See the "OCCURS Clause", Chapter 9). Then, if the AT END phrase is specified, imperative-statement-1 is executed; if the AT END phrase is not specified, control passes to the end of the SEARCH statement.

   b. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is not greater than the highest permissible occurrence number for identifier-1 (the number of occurrences of identifier-1, the last of which is the highest permissible is discussed in the OCCURS clause) the SEARCH statement operates by evaluating the conditions in the order that they are written, making use of the index settings, wherever specified, to determine the occurrence of those items to be tested. If none of the conditions is satisfied, the index-name for identifier-1 is incremented to obtain reference to the next occurrence. The process is then repeated using the new index-name settings unless the new value of the index-name settings for identifier-1 corresponds to a table element outside the permissible range of occurrence values, in which case the search terminates as indicated in 2a above. If one of the conditions is satisfied upon its evaluation, the search terminates immediately and control passes to the imperative statement associated with that condition, if present, or, if the NEXT SENTENCE phrase is associated with that condition, to the next executable sentence; the index-name remains set at the occurrence which caused the condition to be satisfied.

3. In a Format 2 SEARCH statement, the results of the SEARCH ALL operation are predictable only when:

   a. The data in the table is ordered in the same manner as described in the KEY IS phrase of the OCCURS clause referenced by identifier-1, and

   b. The contents of the key(s) referenced in the WHEN phrase are sufficient to identify a unique table element.

4. If Format 2 of the SEARCH statement is used, a non-serial type of search operation may take place; the initial setting of the index-name for identifier-1 is ignored and its setting is varied during the search operation in a manner such that at no time is it set to a value that exceeds the value which corresponds to the last element of the table, or that is less than the value that corresponds to the first element of the table. The length of the table is discussed in the OCCURS clause (See Chapter 9). The OCCURS Clause). If any of the conditions specified in the WHEN phrase cannot be satisfied for any setting of the index within the permitted range, control is passed to imperative-statement-1 of the AT END phrase, when specified, or to the end of the SEARCH statement when this phrase is not specified; in either case the final setting of the index is not predictable. If all the conditions can be satisfied, the index indicates an occurrence that allows the conditions to be satisfied, and control passes to imperative-statement-2, if specified, or to the next executable sentence if the NEXT SENTENCE phrase is specified.

5. After execution of imperative-statement-1 or imperative-statement-2, that does not terminate with a GO TO statement, control passes to the end of the SEARCH statement.

6. In Format 2, the index-name that is used for the search operation is the first (or only) index-name specified in the INDEXED BY phrase in the OCCURS clause referenced by identifier-1. Any other index-names for identifier-1 remain unchanged.

7. In Format 1, if the VARYING phrase is not used, the index-name that is used for the search operation is the first (or only) index-name specified in the INDEXED BY phrase in the OCCURS clause referenced by identifier-1. Any other index-names for identifier-1 remain unchanged.

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

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

    b. If the VARYING identifier-2 phrase is specified, and identifier-2 is an index data item, then the data item referenced by identifier-2 is incremented by the same amount as, and at the same time as, the index associated with identifier-1 is incremented. If identifier-2 is not an index data item, the data item referenced by identifier-2 is incremented by the value one (1) at the same time as the index referenced by the index-name associated with identifier-1 is incremented.

9. If identifier-1 references a data item subordinate to a data item that contains an OCCURS clause, an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with identifier-1 (and the data item identifier-2 or index-name-1, if present) is modified by the execution of the SEARCH statement. To search a multi-dimensional table it is necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed whenever index-names must be adjusted to appropriate settings.

10. A description of the function of the END-SEARCH phrase is given in the appropriate paragraph. (See "Scope of Statements" in Chapter 10.)

13.2 SEND

Description

The SEND statement causes a message, a message segment, or a portion of a message or segment to be released to one or more to output queues maintained by the Message Control System (MCS).

Format 1

```
SEND cd-name-1 FROM identifier-1
```

Format 2

```
                                        {WITH identifier-2}
SEND cd-name-1 [FROM identifier-1] {WITH ESI         }
                                        {WITH EGI         }
                                        {WITH EMI         }

                          {{identifier-3} [LINE ]}
     {BEFORE}             {{integer-1   } [LINES]}
    [{      } ADVANCING {                       }]
     {AFTER }            { mnemonic-name-1      }
                         { PAGE                 }

    [REPLACING LINE]
```

Syntax Rules

1. Cd-name-1 must reference an output CD or input-output CD.
2. Identifier-2 must reference a one character integer without an operational sign.
3. Identifier-3 must reference an integer data item.
4. Mnemonic-name-1, if specified, must correspond to the word "PAGE".
5. Integer-1 or the value of the data item referenced by identifier-3 may be zero.
6. If identifier-1 is a function-identifier, it must reference an alphanumeric function.

General Rules

All Formats

1. When a receiving communications device (printer, display screen, card punch, etc.) is oriented to a fixed line size:

   a. Each message or message segment begins at the leftmost character position of the physical line.

   b. A message or message segment that is smaller than the physical line size is released so as to appear space filled to the right.

   c. Excess characters of a message or message segment are not truncated. Characters are packed to a size equal to that of the physical line and then transmitted to the output device. The process continues on the next line with the excess characters.

2. When a receiving communication device (paper tape punch, another computer, etc.) is oriented to handle variable length messages, each message or message segment will begin on the next available character position of the communications device.

3. As part of the execution of a SEND statement, the MCS will interpret the content of the TEXT LENGTH data item of the area referenced by cd-name-1 to be the user's indication of the number of leftmost character positions of the data item referenced by identifier-1 from which data is to be transferred. (See the "Communication Description - Complete Entry Skeleton" in Chapter 8.)

   If the content of the TEXT LENGTH data item of the area referenced by cd-name-1 is zero, no characters of the data item referenced by identifier-1 are transferred.

   If the content of the TEXT LENGTH data item of the area referenced by cd-name-1 is outside the range of zero through the size of the data item referenced by identifier-1 inclusive, an error is indicated by the value of the STATUS KEY data item of the area referenced by cd-name-1, and no data is transferred (see "Communication Status Key Conditions", Chapter 8).

4. As part of the execution of a SEND statement, the content of the STATUS KEY data item of the area referenced by cd-name-1 is updated by the MCS (see the "Communication Description", Chapter 8).

5. The effect of having special control characters within the content of the data item referenced by identifier-1 is undefined.

6. A single execution of a Format 1 SEND statement releases only a single portion of message segment or a single portion of a message to the MCS.

   A single execution of a Format 2 SEND statement never releases to the MCS more than a single message or a single message segment as indicated by the content of the data item referenced by identifier-2 or by the specified indicator ESI, EMI, or EGI.

   However, the MCS will not transmit any portion of a message to a communications device until the entire message has been released to the MCS.

7. During the execution of the run unit, the disposition of a portion of a message which is not terminated by an EMI or EGI or which has not been eliminated by the execution of a PURGE statement is undefined. However, the message does not logically exist for the MCS and hence cannot be sent to a destination.

   After the execution of a STOP RUN statement, any portion of a message transferred from the run unit via a SEND statement, but not terminated by an EMI or EGI is purged from the system. Thus no portion of the message is sent.

8. Once the execution of a SEND statement has released a portion of a message to the MCS, only subsequent execution of SEND statements in the same run unit can cause the remaining portion of the message to be released.

Format 2

9. The content of the data item referenced by identifier-2 indicates that the content of the data item referenced by identifier-1, when specified, is to have an associated End of Message Indicator, End of Group Indicator, or no indicator (which implies a portion of a message or a portion of a segment). If identifier-1 is not specified, only the indicator is transmitted to the MCS.

If the content if the data-item referenced by identifier-2 is	then the content data item referenced by identifier-1 has an associated	which means
'0'	no indicator	Portion of a message or of a segment
'1'	End of Segment Indicator (ESI)	End of current segment
'2'	End of Message Indicator (EMI)	End of current message
'3'	End of Group Indicator (EGI)	End of the current group of messages

Any character other than '1', '2', or '3' will be interpreted as '0'.

If the content of the data item referenced by identifier-2 is other than '1', '2', or '3', and identifier-1 is not specified, then an error is indicated by the value in the STATUS KEY data item of the area referenced by cd-name-1, and no data is transferred.

10. The WITH EMI phrase indicates to the MCS that the message is complete.

    The WITH EGI phrase indicates to the MCS that the group of messages is complete.

    The WITH ESI phrase indicates to the MCS that the message segment is complete.

    The MCS will recognize these indications and establish whatever is necessary to maintain segment, message, or group control.

11. The hierarchy of ending indicators is EGI, EMI, and ESI. An EGI need not be preceded by an ESI or EMI. An EMI need not be preceded by an ESI. An EGI need not be preceded by an EMI.

NOTE: For more explanation, see the *MCS User's Guide*.

12. The ADVANCING phrase allows control of the vertical positioning of each message or message segment on a communication device where vertical positioning is applicable. If vertical positioning is not applicable on the device, the MCS will ignore the vertical positioning specified or implied.

13. If identifier-2 is specified and the content of the data item referenced by identifier-2 is zero, the ADVANCING phrase and the REPLACING phrase, if specified, are ignored by the MCS.

14. On a device where vertical positioning is applicable and the ADVANCING phrase is not specified, automatic advancing occurs as if the user had specified AFTER ADVANCING 1 LINE.

15. If the ADVANCING phrase is implicitly or explicitly specified and vertical positioning is applicable, the following rules apply:

    a. If integer-1 or identifier-3 is specified, characters transmitted to the communication device are repositioned vertically downward the number of lines equal to integer-1 or the value of the data item referenced by identifier-3.

    b. If the value of the data item referenced by identifier-3 is negative, the results are undefined.

    c. If the BEFORE phrase is used, the message or message segment is represented on the communication device before vertical positioning according to General Rule 15a above.

    d. IF the AFTER phrase is used, the message or message segment is represented on the communication device after vertical positioning according to General Rule 15a above.

    e. If PAGE is specified, characters transmitted to the communication device are represented on the device before or after (depending upon the phrase used) the device is repositioned to the next (new) page. If PAGE is specified but page has no meaning in conjunction with a specific device, then advancing occurs as if the user had specified BEFORE or AFTER (depending upon the phrase used) ADVANCING 1 LINE.

16. When a receiving communication device is a character-imaging device on which it is possible to present two or more characters at the same position and the device permits the choice of either the second or subsequent characters appearing superimposed on characters already displayed at that position or each character appearing in the place of the characters previously transmitted to that line:

    a. If the REPLACING phrase is specified, the characters transmitted by the SEND statement replace all characters which may have previously been transmitted to the same line beginning with the leftmost character position of the line.

    b. If the REPLACING phrase is not specified, the characters transmitted by the SEND statement appear superimposed upon the characters which may have previously been transmitted to the same line beginning with the leftmost character position of the line.

17. When a receiving communication device does not support the replacement of characters, regardless of whether or not the REPLACING phrase is specified, the characters transmitted by the SEND statement appear superimposed upon the characters which may have previously been transmitted to the same line, beginning with the leftmost character position of the line.

18. When a receiving communication device does not support the super-imposition of two or more characters at the same position, regardless of whether or not the REPLACING phrase is specified, the characters transmitted by the SEND statement replace all characters which may have previously been transmitted to the same line beginning with the leftmost character position of the line.

13.3 SET

Description

1. The SET statement establishes reference points for table handling operations by setting indices associated with table elements.

2. The SET statement is also used to alter the status of external switches.

3. The SET statement is also used to alter the value of conditional variables.

4. The SET statement is also used to store a value in a pointer data-item or associate an address with a data item declared in linkage section.

Format 1

```
    {identifier-1}        {identifier-2}
SET {             } ... TO {index-name-2}
    {index-name-1}        {integer-1   }
```

Format 2

```
                        {UP BY  } {identifier-3}
SET {index-name-3}... {       } {             }
                        {DOWN BY} {integer-2   }
```

Format 3

```
     {SWITCH-n       }       {ON }
SET {{               }... TO {   }}...
     {mnemonic-name-1}       {OFF}
```

Format 4

```
                              {   TRUE    }
SET {condition-name}... TO    { |-------| }
                              { | FALSE | }
                                |-------|
```

Format 5

```
|-------------------------------------------------------------|
|     {identifier-4          }       {identifier-6           } |
| SET {                      }... TO {ADDRESS OF identifier-7} |
|     {ADDRESS OF data-name-1}       {NULL                   } |
|-------------------------------------------------------------|
```

Syntax Rules

1. When SWITCH-n is specified, n must be an unsigned integer ranging from 0 to 31 and written without leading zeros.

2. Identifier-1 and identifier-2 must each reference an index data item or an elementary item described as an integer.

3. Identifier-3 must reference an elementary numeric integer.

4. Integer-1 and integer-2 may be signed. Integer-1 must be positive.

5. Each mnemonic-name must be associated with an external switch (SWITCH-n), the status of which may be altered (see the "SPECIAL-NAMES Paragraph" in Chapter 7).

6. Condition-name must be associated with a conditional variable.

|7. In format 4, if the FALSE phrase is specified, the FALSE phrase must be specified in the VALUE clause of the Data Description entry for Condition-name.

8. identifier-4 and identifier-6 must be described with USAGE IS POINTER clause.

9. Data-name-1 must be level 01 or 77 item defined in the linkage section. It must not appear in the USING phrase of the PROCEDURE DIVISION. It must not be subject of REDEFINES.

10. Identifier-7 may be subscripted or reference modified.|

General Rules

Formats 1 and 2

1. Index-names are associated with a given table by being specified in the INDEXED BY phrase of the OCCURS clause for that table.

2. If index-name-1 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 table associated with index-name-1. The value of the index associated with an index-name after the execution of a PERFORM or SEARCH statement may be set to an occurrence number that is outside the range of its associated table (see the "SEARCH Statement" in Chapter 13, and the "PERFORM Statement" in Chapter 12).

   If index-name-2 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 table associated with index-name-1.

   If index-name-3 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 table associated with index-name-3.

Format 1

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-2, identifier-2, or integer-1. If identifier-2 references an index data item, or if index-name-2 is related to the same table as index-name-1, no conversion takes place.

   b. If identifier-1 references an index data item, it may be set equal to either the content of index-name-2 or identifier-2, where identifier-2 also references an index data item; no conversion takes place in either case.

   c. If identifier-1 does not reference an index data item, it may be set only to an occurrence number that corresponds to the value of index-name-2. Neither identifier-2 nor integer-1 can be used in this case.

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

Format 2

4. In Format 2, the content of index-name-3 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 the data item referenced by identifier-3; thereafter, the process is repeated for each recurrence of index-name-3. For each repetition, the value of the data item referenced by identifier-3 is used as it was at the beginning of execution of the statement.

5. Data in the table below represents the validity of various operand combinations in Formats 1 of the SET statement. The General Rule Reference indicates the applicable General Rule.

Table 13-1. Permissible SET Statement Operands

sending Item	**Receiving Item**					
	Integer Data Item		Index		Index Data Item	
	Permissible	Rule	Permissible	Rule	Permissible	Rule
Integer Literal	no	3c	valid	3a	no	3b
Integer Data Item	no	3c	valid	3a	no	3b
Index	valid	3c	valid	3a	valid	3b+
Index Data Item	no	3c	valid	3a	valid	3b+
+ There is no conversion.						

Format 3

6. The status of each external switch explicitly referenced by its name SWITCH-n or associated with the specified mnemonic-name is modified such that the truth value resultant from evaluation of a condition-name associated with that switch will reflect an "on" status if the ON phrase is specified or an "off" status if the OFF phrase is specified (see "Switch-Status Condition", Chapter 10).

Format 4

7. If the TRUE phrase is specified, the literal in the VALUE clause associated with condition-name1 is placed in the conditional variable according to the rules of the VALUE clause. If more than one literal is specified in the VALUE clause, the conditional variable is set to the value of the first literal that appears in the VALUE clause.

|8. If the FALSE phrase is specified, the literal in the FALSE phrase of the VALUE clause associated with condition-name is placed in the condition variable according to the rules of the VALUE clause.|

9. If multiple condition-names are specified, the results are the same as if a separate SET statement had been written for each condition-name-1 in the same order as specified in the SET statement.

|Format 5

10. The pointer data item referenced by identifier-4 is set to the contents of the pointer data item referenced by identifier-6, if specified, or to the address of the data item referenced by identifier-7, if specified, or to an invalid address (HIGH-VALUE) if the reserved word NULL is specified.

11. The setting of ADDRESS OF data-name-1 causes the data item referenced by data-name-1 to occupy the same storage area as the data item whose address was previously set in the pointer data item referenced by identifier-6, if specified, or as the data item referenced by identifier-7, if specified. If the reserved word NULL is specified, or if the data item referenced by identifier-6 contains an invalid address, the result of any subsequent reference that needs to access the contents of data-name-1 will be unpredictable.

12. If identifier-7 is specified and it does not reference an external data item, and if the program in which this data item is declared is a called program, the address of identifier-7 as computed by the SET statement becomes invalid when control returns to the calling program.|

13.4 SORT

Description

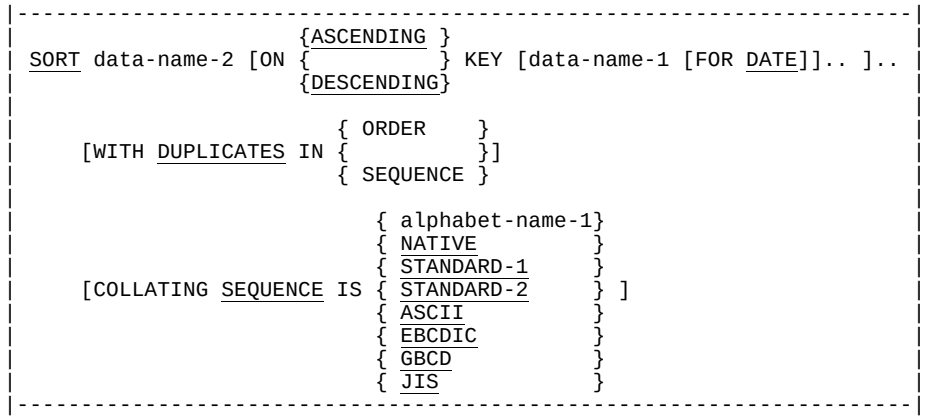
The SORT statement creates a sort file by executing an input procedure or by transferring records from another file, sorts the records in the sort file on a set of specified keys; and in the final phase of the sort operation, makes available each record from the sort file, in sorted order, to an output procedure or to an output file.

The format 2 allows to sort a table.

Format 1

```
                      {ASCENDING }
SORT file-name-1 {ON [{          } KEY {data-name-1 [FOR DATE]}... }...
                      {DESCENDING}

                         {   ORDER      }
    [WITH DUPLICATES IN  { |----------| }]
                         { | SEQUENCE | }
                           |----------|

                           {   alphabet-name-1}
                           { |------------|   }
                           { | NATIVE     |   }
                           { | STANDARD-1 |   }
    [COLLATING SEQUENCE IS { | STANDARD-2 |   }  ]
                           { | ASCII      |   }
                           { | EBCDIC     |   }
                           { | GBCD       |   }
                           { | JIS        |   }
                             |------------|

    {INPUT PROCEDURE IS                                  }
    {                       {THROUGH}                    }
    {      procedure-name-1 [{       } procedure-name-2]}
    {                       {THRU   }                    }
    {                                                    }
    {USING {file-name-2}...                              }

    {OUTPUT PROCEDURE IS                                 }
    {                       {THROUGH}                    }
    {      procedure-name-3 [{       } procedure-name-4]}
    {                       {THRU   }                    }
    {                                                    }
    {GIVING {file-name-3}...                             }
```

Format 2



Syntax Rules

All Formats

1. A SORT statement may appear anywhere in the Procedure Division except in the declaratives portion or in an input procedure or an output procedure associated with a SORT or a MERGE statement.

|2. The words ORDER and SEQUENCE are equivalent.|

Format 1

3. File-name-1 must be described in a Sort-Merge File description entry in the Data Division.

4. If the USING phrase is specified and the file referenced by file-name-1 contains variable-length records, the size of the records contained in the file referenced by file-name-2 must not be less than the smallest record nor larger than the largest record described for file-name-1. If the file referenced by file-name-1 contains fixed-length records, the size of the records contained in the file referenced by file-name-2 must not be larger than the largest record described for the file referenced by file-name-1.

5. Data-name-1 is a KEY data-name. KEY data-names are subject to the following rules:

   a. The data items identified by KEY data-names must be described in records associated with file-name-1.

   b. KEY data-names may be qualified.

   c. The data items identified by KEY data-names must not be variable length data items, nor may they name group items which contain variable-occurrence data items.

   d. If file-name-1 has more than one record description, then the data items identified by KEY data-names need be described in only one of the record descriptions. The same character positions which are referenced by a key data-name in one record description entry are taken as the key in all records of the file.

   e. None of the data items identified by key data-names can be described by an entry which either contains an OCCURS clause or is subordinate to an entry which contains an OCCURS clause.

   f. If the file referenced by file-name-1 contains variable length records, all the data items identified by key data-names must be contained within the first x character positions of the record, where x equals the minimum record size specified for the file referenced by file-name-1.

   |g. The data items identified by key data-names must not be described as boolean or pointer data items.|

   h. If the FOR DATE phrase is specified, data-name-1 must be described as PIC 99 USAGE DISPLAY.

6. The words THRU and THROUGH are equivalent.

7. file-name-2 and file-name-3 must be described in a file description entry, not in a Sort-Merge File Description entry, in the Data Division.

8. The files referenced by file-name-2 and file-name-3 may reside on the same multiple file reel.

9. If file-name-3 references an indexed file, then the major key must be associated with the ASCENDING phrase and the first data-name must specify the same character positions in its record as are specified for the prime record key for that file.

10. No pair of file-names in the same SORT statement may be specified in the same SAME SORT AREA or SAME SORT-MERGE AREA clause. File-names associated with the GIVING phrase may not be specified in the same SAME AREA clause. (See the "I-O-CONTROL" paragraph in Chapter 7.)

11. If the GIVING phrase is specified and the file referenced by file-name-3 contains variable length records, the size of the records contained in the file referenced by file-name-1 must not be less than the smallest record nor larger than the largest record described for file-name-3. If the file referenced by file-name-3 contains fixed-length records, the size of the records contained in the file referenced by file-name-1 must not be larger than the largest record described for the file referenced by file-name-3.

Format 2

12. Data-name-2 can be qualified and must have an occurs clause in its Data Description entry.

13. The data-item referenced by data-name-1 must be the same as the data-item referenced by data-name-2, or an entry subordinate to the data-item referenced by data-name-2.

14. The data-item referenced by data-name-1 must not be described by the entry containing an OCCURS clause unless it refers to the same data item referenced by data-name-2. The data-item referenced by data-name-1 must not be subordinate to an entry containing an OCCURS clause that is also subordinate to data-name-2.

15. The KEY phrase can be omitted only if the description of the table referenced by data-name-2 contains a KEY phrase.

16. If the FOR DATE phrase is specified, data-name-1 must be described as PIC 99 USAGE DISPLAY.

General Rules

All Formats

1. The data-names following the word KEY are listed from left to right in the SORT statement in order of decreasing significance without regard to how they are divided into KEY phrases. The leftmost data-name is the major key, the next data-name is the next most significant key, etc.

    a. When the ASCENDING phrase is specified, the sorted sequence will be from the lowest value of the contents of the data items identified by the KEY data-names to the highest value, according to the rules for comparison of operands in a relation condition.

    b. When the DESCENDING phrase is specified, the sorted sequence will be from the highest value of the contents of the data items identified by the KEY data-names to the lowest value, according to the rules for comparison of operands in a relation condition.

2. If the DUPLICATES phrase is specified and the contents of all the key data items associated with one data record are equal to the contents of the corresponding key data items associated with one or more other data records, then the order of return of these records is:

    a. the order of the associated input files as specified in the SORT statement. Within a given input file the order is that in which the records are accessed from that file.

    b. the order in which these records are released by an input procedure, when an input procedure is specified.

3. If the DUPLICATES phrase is not specified and the contents of all the key data items associated with one data record are equal to the contents of the corresponding key data items associated with one or more other data records, then the order of return of these records is undefined.

4. The collating sequence that applies to the comparison of the non-numeric key data items specified is determined at the beginning of the execution of the SORT statement in the following order of precedence:

   a. First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in the SORT statement.

   b. Second, the collating sequence established as the program collating sequence.

5. If the FOR DATE phrase is specified the SORT proceeds following the "Rule 61" for the corresponding data-name-1 (refer to *SORT/MERGE Utilities User's Guide*.).

Format 1

5. If the file referenced by file-name-1 contains only fixed-length records, any record in the file referenced by file-name-2 containing fewer character positions than that fixed length is space filled on the right beginning with the first character position after the last character in the record when that record is released to the file referenced by file-name-1.

6. The execution of the SORT statement consists of three distinct phases as follows:

   a. Records are made available to the file referenced by file-name-1. This is achieved either by the execution of RELEASE statements in the input procedure or by the implicit execution of READ statements for file-name-2. When this phase commences, the file referenced by file-name-2 must not be in the open mode. When this phase terminates, the file referenced by file-name-2 is not in the open mode.

   b. The file referenced by file-name-1 is sequenced. No processing of the files referenced by file-name-2 and file-name-3 takes place during this phase.

   c. The records of the file referenced by file-name-1 are made available in sorted order. The sorted records are either written to the file referenced by file-name-3 or, by the execution of a RETURN statement, are made available for processing by the output procedure. When this phase commences, the file referenced by file-name-3 must not be in the open mode. When this phase terminates, the file referenced by file-name-3 is not in the open mode.

7. The input procedure may consist of any procedure needed to select, modify, or copy the records that are made available one at a time by the RELEASE statement to the file referenced by file-name-1. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the input procedure, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the input procedure. The range of the input procedure must not cause the execution of any MERGE, RETURN, or SORT statement.

8. If an input procedure is specified, control is passed to the input procedure before the file referenced by file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last statement in the input procedure and when control passes the last statement in the input procedure, the records that have been released to the file referenced by file-name-1 are sorted.

9. If the USING phrase is specified, all the records in the file(s) referenced by file-name-2 are transferred to the file referenced by file-name-1. For each of the files referenced by file-name-2 the execution of the SORT statement causes the following actions to be taken:

   a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the INPUT phrase had been executed.

   b. The logical records are obtained and released to the sort operation. Each record is obtained as if a READ statement with the NEXT and the AT END phrases had been executed.

      For a relative file, the content of the relative key data item is undefined after the execution of the SORT statement if file-name-2 is not referenced in the GIVING phrase.

   c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed. This termination is performed before the file referenced by file-name-1 is sequenced by the SORT statement.

      These implicit functions are performed such that any associated USE AFTER EXCEPTION/ERROR procedures are executed; however, the execution of such a USE procedure must not cause the execution of any statement manipulating the file referenced by, or accessing the record area associated with, file-name-2.

10. The output procedure may consist of any procedure needed to select, modify, or copy the records that are made available one at a time by the RETURN statement in sorted order from the file referenced by file-name-1. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the output procedure, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the output procedure. The range of the output procedure must not cause the execution of any MERGE, RELEASE, or SORT statement.

11. If an output procedure is specified, control passes to it after the file referenced by file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last statement in the output procedure and when control passes the last statement in the output procedure, the return mechanism provides for termination of the sort and then passes control to the next executable statement after the SORT statement. Before entering the output procedure, the sort procedure reaches a point at which it can select the next record in sorted order when requested. The RETURN statements in the output procedure are the requests for the next record.

12. If the GIVING phrase is specified, all the sorted records are written on the file referenced by file-name-3 as the implied output procedure for the SORT statement. For each of the files referenced by File-name-3, the execution of the SORT statement causes the following actions to be taken:

    a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase has been executed. This initiation is performed after the execution of any input procedure.

    b. The sorted logical records are returned and written onto the file. The records are written as if a WRITE statement without any optional phrases had been executed.

       For a relative file, the relative key item for the first record returned contains the value '1'; for the second record returned, the value '2', etc. After execution of the SORT statement, the content of the relative key data item indicates the last record returned to the file.

    c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

    These implicit functions are performed such that any associated USE AFTER EXCEPTION/ERROR procedures are executed; however, the execution of such a USE procedure must not cause the execution of any statement manipulating the file referenced by, or accessing the record area associated with, file-name-3. On the first attempt to write beyond the externally defined boundaries of the file, any USE AFTER STANDARD EXCEPTION/ERROR procedure specified for the file is executed; if control is returned from that USE procedure or if no such USE procedure is specified, the processing of the file is terminated as in paragraph 12c above.

13. If the file referenced by file-name-3 contains only fixed length records, any record in the file referenced by file-name-1 containing fewer character positions than that fixed length is space filled on the right beginning with the first character position after the last character in the record when that record is returned to the file referenced by file-name-3.

14. Segmentation can be applied to programs containing the SORT statement. However, the following restrictions apply:

    If a sort statement appears in a section that is not in an independent segment, then any input procedures or output procedures referenced by that SORT statement must appear totally within non independent segments, or wholly contained in a single independent segment.

    If a SORT statement appears in an independent segment, then any input procedures or output procedures referenced by that SORT statement must be contained totally within non independent segments, or wholly within the same independent segment as that SORT statement.

Format 2

15. The SORT statement sorts the table referenced by data-name-2 and presents the sorted table in data-name-2 either in the order determined by the ASCENDING or DESCENDING phrases, if specified, or in the order determined by the KEY phrase associated with data-name-2.

16. To determine the relative order in which the table elements are stored after sorting, the contents of corresponding key data items are compared according to the rules for comparison of operands in a relation condition, starting with the most significant key data item.

    a. If the contents of the corresponding key data items are not equal and the key is associated with the ASCENDING phrase, the table element containing the key data item with the lower value has the lowest occurrence number;

    b. If the contents of the corresponding key data items are not equal and the key is associated with the DESCENDING phrase, the table element containing the key data item with the higher value has the lowest occurrence number; and

    c. If the contents of the corresponding key data items are equal, the determination is based on the contents of the next most significant key data item.

17. The number of occurrences of table elements referenced by data-name-2 is determined by the rules in the OCCURS clause.

18. If the KEY phrase is not specified, the sequence is determined by the KEY phrase in the Data Description entry of the table referenced by data-name-2.

19. If the KEY phrase is specified, it overrides any KEY phrase specified in the Data Description entry of the table referenced by data-name-2.

20. If data-name-1 is omitted, the data item referenced by data-name-2 is the key data item.

13.5 START

Description

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

Format -1

```
START file-name-1

           {   IS EQUAL TO                }
           {   IS =                       }
           { |---------|                  }
           { | EQUALS  |                  }
           { | EXCEEDS |                  }
           { |---------|                  }
     [KEY  {   IS GREATER THAN            } data-name-1]
           {   IS >                       }
           {   IS NOT LESS THAN           }
           {   IS NOT <                   }
           {   IS GREATER THAN OR EQUAL TO}
           {   IS >=                      }

     [INVALID KEY imperative-statement-1]

     [NOT INVALID KEY imperative-statement-2]

     [END-START]
```

Format -2

```
|-----------------------------------------------------------|
|                                                           |
|  START file-name-1                                        |
|           {                                 }             |
|           {   IS LESS THAN                  }             |
|           {   IS <                          }             |
|       KEY {   IS NOT GREATER THAN           } data-name-1 |
|           {   IS NOT >                      }             |
|           {   IS LESS THAN OR EQUAL TO      }             |
|           {   IS <=                         }             |
|                                                           |
|      [INVALID KEY imperative-statement-1]                 |
|                                                           |
|      [NOT INVALID KEY imperative-statement-2]             |
|                                                           |
|      [END-START]                                          |
|-----------------------------------------------------------|
```

Syntax Rules

1. File-name-1 must be the name of a relative or indexed file, if Format 1 is used |and the name of an indexed file, if Format 2 is used.|

2. File-name-1 must not be the name of a sort or merge file.

3. Data-name-1 may be qualified.

4. The INVALID KEY phrase must be specified if no applicable USE AFTER STANDARD EXCEPTION procedure is specified for file-name-1.

5. If the file referenced by file-name-1 is a relative file, data-name-1, if specified, must be the data item specified in the RELATIVE KEY phrase in the ACCESS MODE clause of the associated File Description entry.

6. If file-name-1 is the name of an indexed file, and if the KEY phrase is specified, data-name-1 must reference either:

   a. A data item specified as a record key associated with file-name-1, or

   b. Any data item of category alphanumeric whose leftmost character position within a record of the file corresponds to the leftmost character position of a record key associated with file-name-1 and whose length is not greater than the length of that record key.

General Rules

All Files

1. The file referenced by file-name-1 must be open in the INPUT or I-O mode at the time that the START statement is executed (see the "OPEN Statement" in Chapter 12).

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

3. The execution of the START statement does not alter either the content of the record area, or the content of the data item referenced by the data-name specified in the DEPENDING ON phrase of the RECORD clause associated with file-name-1.

4. 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-1 and a data item as specified in General Rules 10, 12, and 13.
   If file-name-1 references an indexed file, the comparison is made on the ascending key of reference according to the collating sequence of the file.
   If file-name-1 references an indexed file and the operands are of unequal size, comparison proceeds as though the longer one was truncated on the right such that its length is equal to that of the shorter.

All other numeric or non-numeric comparison rules apply (see "Comparison of Numeric Operands" and "Comparison of Non-numeric Operands", Chapter 10).

a. For relative files, file position indicator is set to the relative record number of the first logical record in the file whose key satisfies the comparison.
For indexed files, the file position indicator is set to the value of the key of reference in the first logical record , |or last logical record if Format 2 is used| whose key satisfies the comparison.

b. If the comparison is not satisfied by any record in the file, the invalid key condition exists and the execution of the START statement is unsuccessful. (See the "Invalid Key Condition" in Chapter 10.)

5. The execution of the START statement causes the value of the I-O status associated with file-name-1 to be updated (see "I-O Status", Chapter 7).

6. If, at the time of the execution of the START statement, the file position indicator indicates that an optional input file is not present, the invalid key condition exists and the execution of the START statement is unsuccessful. (See the "Invalid Key Condition" in Chapter 10.)

7. Transfer of control following the successful or unsuccessful execution of the START operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the START statement. (see the "Invalid Key Condition" in Chapter 10).

8. Following the unsuccessful execution of a START statement, the file position indicator is set to indicate that no valid next record has been established. Also, for indexed files, the key of reference is undefined.

9. The END-START phrase delimits the scope of the START statement. A description of the function of the END-START phrase is given in the appropriate paragraph. (See "Scope of Statements" in Chapter 10.)

Relative Files:

10. The comparison described in General Rule 4 uses the data item referenced by the RELATIVE KEY phrase of the ACCESS MODE clause associated with file-name-1.

Indexed Files

11. A Key of Reference is established as follows:

    a. If the KEY phrase is not specified, the Prime Record Key specified for file-name-1 becomes the Key of Reference.

    b. If the KEY phrase is specified, and data-name-1 is specified as a Record Key for file-name-1, that Record Key becomes the Key of Reference.

    c. If the KEY phrase is specified, and data-name-1 is not specified as a record key for file-name-1, the record key whose leftmost character position corresponds to the leftmost character position of the data item specified by data-name-1, becomes the key of reference.

    This Key of Reference is used to establish the ordering of records for the purpose of this START statement, see "General Rule" 4; and, if the execution of the START is successful, the Key of Reference is also used for subsequent sequential READ statements. (See the "READ Statement" in Chapter 12.)

12. If the KEY phrase is specified, the comparison described in General Rule 4 uses the data item referenced by data-name-1.

13. If the KEY phrase is not specified, the comparison described in General Rule 4 uses the data item referenced in the RECORD KEY clause associated with file-name-1.

13.6 STOP

Description

The STOP statement causes a permanent or temporary suspension of the execution of the run unit. The literal variation of the STOP statement is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

Format

```
      {   RUN     }
      {   literal }
STOP  { |-------| }
      { | ERROR | }
        |-------|
```

Syntax Rules

1. Literal may be any figurative constant except ALL literal.

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

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

General Rules

1. If the RUN phrase is specified, execution of the run unit ceases and control is transferred to the operating system.

2. During the execution of a STOP RUN statement, an implicit CLOSE statement without any optional phrases is executed for each file that is in the open mode in the run unit. Any USE procedures associated with any of these files are not executed.

3. If the run unit has been accessing messages, the STOP RUN statement causes the message control system (MCS) to eliminate from the queue any message partially received by that run unit. Any portion of a message transferred from the run unit via a SEND statement, but not terminated by and EMI or EGI, is purged from the system.

4. If STOP literal statement is specified, the literal is communicated to the operator and execution of statements in the run unit is temporarily suspended. When execution of statements in the run unit resumes, such execution begins with the next executable statement after the STOP literal statement.

|5. The ERROR phrase has been implemented for debugging purpose. If used, the execution of the run unit aborts and control is transferred to the operating system. It is the way, in case of malfunction, to get a dump where all elements (hardware registers, etc.) remain unchanged.|

13.7 STRING

Description

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

Format

```
        {identifier-1}                 {identifier-2}
STRING {{           }... DELIMITED BY {literal-2   }}...
        {literal-1  }                 {SIZE        }

       INTO identifier-3

      [WITH POINTER identifier-4]

      [ON OVERFLOW imperative-statement-1]

      [NOT ON OVERFLOW imperative-statement-2]

      [END-STRING]
```

Syntax Rules

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

2. All literals must be described as non-numeric literals, and all identifiers, except identifier-4, must be described implicitly or explicitly as USAGE IS DISPLAY.

3. Identifier-3 must not be reference modified.

4. Identifier-3 must not represent an edited data item and must not be described with the JUSTIFIED clause.

5. Identifier-4 must be described as an elementary numeric integer data item of sufficient size to contain a value equal to 1 plus the size of the data item referenced by identifier-3. The symbol 'P' may not be used in the PICTURE character-string of identifier-4.

6. Where identifier-1 or identifier-2 is an elementary numeric data item, it must be described as an integer without the symbol 'P' in its PICTURE character-string.

7. None of the identifiers may reference boolean data items.

General Rules

1. Identifier-1 or literal-1, represents the sending items. Identifier-3 represents the receiving item.

2. Literal-2 or the content of the data item referenced by identifier-2 indicates the character(s) delimiting the move. If the SIZE phrase is used, the content of the complete data item defined by identifier-1 or literal-1, is moved. When a figurative constant is used as the delimiter, it is a single character non-numeric literal.

3. When a figurative constant is specified as literal-1 or literal-2, it refers to an implicit one character data item whose USAGE IS DISPLAY.

4. When the STRING statement is executed, the transfer of data is governed by the following rules:

   a. Those characters from literal-1, or from the content of the data item referenced by identifier-1, are transferred to the data item referenced by identifier-3 in accordance with the rules for alphanumeric to alphanumeric moves, except that no space-filling will be provided. (See the "MOVE Statement", Chapter 12).

   b. If the DELIMITED phrase is specified without the SIZE phrase, the content of the data item referenced by identifier-1, or the value of literal-1, is transferred to the receiving data item in the sequence specified in the STRING statement beginning with the leftmost character and continuing from left to right until the end of the sending data item is reached or the end of the receiving data item is reached or until the character(s) specified by literal-2, or by the content of identifier-2 are encountered. The character(s) specified by literal-2 or by the data item referenced by identifier-2 are not transferred.

   c. If the DELIMITED phrase is specified with the SIZE phrase, the entire content of literal-1, or the content of the data item referenced by identifier-1, is transferred in the sequence specified in the STRING statement, to the data item referenced by identifier-3 until all data has been transferred or the end of the data item referenced by identifier-3 has been reached.

   This behavior is repeated until all occurrences literal-1, or data items referenced by identifier-1, have been processed.

5. If the POINTER phrase is specified, the data item referenced by identifier-4 must be set to an initial value greater than zero prior to the execution of the STRING statement.

6. If the POINTER phrase is not specified, the following general rules apply as if the user had specified identifier-4 referencing a data item with an initial value of 1.

7. When characters are transferred to the data item referenced by identifier-3, the moves behave as though the characters were moved one at a time from the source into the character positions of the data item referenced by identifier-3 designated by the value of the data item referenced by identifier-4 (provided the value of the data item referenced by identifier-4 does not exceed the length of the data item referenced by identifier-3), and then the data item referenced by identifier-4 was increased by one prior to the move of the next character or prior to the end of execution of the STRING statement. The value of the data item referenced by identifier-4 is changed during execution of the STRING statement only by the behavior specified above.

8. At the end of execution of the STRING statement, only the portion of the data item referenced by identifier-3 that was referenced during the execution of the STRING statement is changed. All other portions of the data item referenced by identifier-3 will contain data that was present before this execution of the STRING statement.

9. Before each move of a character to the data item referenced by identifier-3, if the value associated with the data item referenced by identifier-4 is either less than one or exceeds the number of character positions in the data item referenced by identifier-3, no (further) data is transferred to the data item referenced by identifier-3, and the NOT ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the STRING statement or, if the ON OVERFLOW phrase is specified, to imperative-statement-1. If control is transferred to imperative-statement-1, execution continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the STRING statement.

10. If, at the time of execution of a STRING statement with the NOT ON OVERFLOW phrase, the conditions described in general rule 9 are not encountered, after completion of the transfer of data according to the other general rules, the ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the STRING statement or, if the NOT ON OVERFLOW phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the STRING statement.

11. The END-STRING phrase delimits the scope of the STRING statement.

12. If identifier-1, or identifier-2, occupies the same storage area as identifier-3, or identifier-4, or if identifier-3 and identifier-4 occupy the same storage area, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See "Overlapping Operands" in Chapter 10.)

13.8 SUBTRACT

Description

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.

Format 1

```
          {identifier-1}
SUBTRACT  {            }... FROM {identifier-2 [ROUNDED]}...
          {literal-1   }

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-SUBTRACT]
```

Format 2

```
          {identifier-1}          {identifier-2}
SUBTRACT  {            }... FROM {            }
          {literal-1   }          {literal-2   }

           GIVING {identifier-3 [ROUNDED]}...

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-SUBTRACT]
```

Format 3

```
          {CORRESPONDING}
SUBTRACT  {             } identifier-1 FROM identifier-2 [ROUNDED]
          {CORR         }

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-SUBTRACT]
```

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 on an elementary numeric edited item. In Format 3, each identifier must refer to a group item.

2. Each literal must be a numeric literal.

3. The composite of operands must not contain more than 18 digits |(up to 30 if the compiler is run with the LEVEL=NSTD parameter).| In format 1 the composite of operands is determined by using all of the operands in a given statement. 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. In format 3 the composite of operands is determined separately for each pair of corresponding data items.

4. CORR is an abbreviation for CORRESPONDING.

General Rules

1. In Format 1, the values of the operands preceding the word FORM are added together and the sum is stored in a temporary data item. The value of this temporary data item is subtracted from the value of the data item referenced by identifier-2, storing the result into the data item referenced by identifier-2, and repeating this process for each occurrence of identifier-2 in the left-to-right order in which identifier-2 is specified.

2. In format 2, all literals and the values of the data items referenced by the identifiers preceding the word FROM are added together, the sum is subtracted from literal-2 or the value of the data item referenced by identifier-2 and the result of the subtraction is stored as the new content of each data item referenced by identifier-3.

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

4. Additional rules and explanation relative to this statement are given under the appropriate paragraph (see the "ROUNDED Option", the "SIZE ERROR Condition", the "CORRESPONDING Option", the "Arithmetic Statements", "Overlapping Operands", and "Multiple Results in Arithmetic Statements", Chapter 10).

13.9 SUPPRESS

Description

The SUPPRESS statement causes the Report Writer Control System (RWCS) to inhibit the presentation of a report group.

Format

```
SUPPRESS PRINTING
```

SYNTAX RULE

The SUPPRESS statement may only appear in a USE BEFORE REPORTING procedure.

General Rules

1. The SUPPRESS statement inhibits presentation only for the report group named in the USE procedure within which the SUPPRESS statement appears.

2. The SUPPRESS statement must be executed each time the presentation of the report group is to be inhibited.

3. When the SUPPRESS statement is executed, the RWCS is instructed to inhibit the processing of the following report group functions:

   a. The presentation of the print lines of the report group,

   b. The processing of all LINE clauses in the report group,

   c. The processing of the NEXT GROUP clause in the report group,

   d. The adjustment of LINE-COUNTER.

13.10 TERMINATE

Description

The TERMINATE statement causes the Report Writer Control System (RWCS) to complete the processing of the specified reports.

Format

```
TERMINATE {report-name-1}...
```

Syntax Rule

Each report-name given in a TERMINATE statement must be defined by an RD entry in the Report Section of the Data Division.

General Rules

1. The TERMINATE statement causes the RWCS to produce all the CONTROL FOOTING report groups beginning with the minor CONTROL FOOTING report group. Then the REPORT FOOTING report group is produced. The RWCS makes the prior set of control data item values available to the CONTROL FOOTING and REPORT FOOTING SOURCE clauses and USE procedures, as though a control break has been sensed in the most major control data-name.

2. If no GENERATE statements have been executed for a report during the interval between the execution of an INITIATE statement and a TERMINATE statement, for that report, the TERMINATE statement does not cause the RWCS to produce any report groups or perform any of the related processing.

3. During report presentation, an automatic function of the RWCS is to process PAGE HEADING and PAGE FOOTING report groups, if defined, when the RWCS must advance the report to a new page for the purpose of presenting a body group (see the table "Body Group Presentation Rules" in Chapter 8).

4. The TERMINATE statement cannot be executed for a report unless the TERMINATE statement was chronologically preceded by an INITIATE statement for that report and for which no TERMINATE statement has yet been executed.

5. If more than one report-name is specified in a TERMINATE statement, the result of executing this TERMINATE statement is the same as if a separate TERMINATE statement had been written for each report-name in the same order as specified in the TERMINATE statement.

6. The TERMINATE statement does not close the file to which the report is assigned; a CLOSE statement for the file must be executed. Every report that is in an initiated condition must be terminated before a CLOSE statement is executed for the associated file.

13.11 TRANSFORM

Description

|The TRANSFORM statement replaces specified characters in a data field according to a transformation rule.|

Format

```
|-------------------------------------------------------------|
|                                                             |
| TRANSFORM identifier-1 CHARACTERS                           |
|                                                             |
|           {figurative-constant-1}    {figurative-constant-2} |
|      FROM {literal-1              } TO {literal-2            } |
|           {identifier-2           }    {identifier-3         } |
|                                                             |
|-------------------------------------------------------------|
```

Syntax Rules

|1. Identifier-1 must not be a numeric data item.

2. The combination of the FROM and TO options determines the transformation rule.

3. Literal-1 and literal-2 must be non-numeric literals.

4. Identifier-2 and identifier-3 must be elementary alphabetic or alphanumeric items, or fixed length group items.

5. Identifier-2 and identifier-3 must not exceed 255 characters in length.

6. A character may not be repeated in literal-1 or identifier-2. If a character is repeated, the results are unpredictable.

7. An uppercase character is not equivalent to the corresponding lower-case character.

8. When figurative-constant-1 or figurative-constant-2 is used, a single instance of the character is implied.|

General Rules

|1. The combination of the FROM and TO options determines the transformation rule as specified in General Rules 2 thru 10, below.

2. If FROM figurative-constant-1 TO figurative-constant-2 is specified, then all characters in the data item represented by identifier-1 equal to the single character figurative-constant-1 are replaced by the single character figurative-constant-2.

3. If FROM figurative-constant-1 TO literal-1 is specified, then all characters in the data item represented by identifier-1 equal to the single character figurative-constant-1 are replaced by the single character literal-2.

4. If FROM figurative-constant-1 TO identifier-3 is specified, then all characters in the data item represented by identifier-1 equal to the single character figurative-constant-1 are replaced by the single character represented by identifier-3.

5. If FROM literal-1 TO figurative-constant-2 is specified, then, all characters in the data item represented by identifier-1 equal to any character in literal-1 are replaced by the single character figurative-constant-2.

6. If FROM literal-1 TO literal-2 is specified, literal-1 and literal-2 must be equal in length or literal-2 must be a single character, then:

    a. If literal-1 and literal-2 are equal in length, any character in the data item represented by identifier-1 equal to a character in literal-1 is replaced by the character in the corresponding position of literal-2.

    b. If the length of literal-2 is one, all characters in the data item represented by identifier-1 that are equal to any character in literal-1 are replaced by the single character in literal-2.

7. If FROM literal-1 TO identifier-3 is specified, literal-1 and identifier-3 must be equal in length or identifier-3 must be a single character, then:

    a. If literal-1 and identifier-3 are equal in length, any character in the data item represented by identifier-1 equal to a character in literal-1 is replaced by the character in the corresponding position of the data item represented by identifier-3.

    b. If the length of identifier-3 is one, any character in the data item represented by identifier-1 equal to any character in literal-1 is replaced by the single character of the data item represented by identifier-3.

8. If FROM identifier-2 TO figurative-constant-2 is specified, then all characters in the data item represented by identifier-1 equal to any character in the data item represented by identifier-2 is replaced by the single character figurative-constant-2.

9. If FROM identifier-2 TO literal-2 is specified, identifier-2 and literal-2 must be equal in length or literal-2 must have a length of one, then:

   a. If identifier-2 and literal-2 are equal in length, then any character in the data item represented by identifier-1 equal to any character in the data item represented by identifier-2 is replaced by the character in the corresponding position of literal-2.

   b. If the length of literal-2 is one, then all characters in the data item represented by identifier-1 equal to any character in the data item represented by identifier-2 are replaced by the single character literal-2.

10. If FROM identifier-2 TO identifier-3 is specified, then identifier-2 and identifier-3 must be of equal length or identifier-3 must have a length of one, then:

    a. If identifier-2 and identifier-3 are equal in length, then any character in the data item represented by identifier-1 equal to any character in the data item represented by identifier-2 is replaced by the character in the corresponding position of the data item represented by identifier-3.

    b. If the length of identifier-3 is one, then all characters in the data item represented by identifier-1 equal to any character in the data item represented by identifier-2 are replaced by the single character in the data item represented by identifier-3.|

13.12 UNSTRING

Description

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

Format

```
UNSTRING identifier-1

                          {identifier-2}
       [DELIMITED BY [ALL] {            }
                          {literal-1    }

                          {identifier-3}
                 [OR [ALL] {            }]... ]
                          {literal-2    }

        INTO {identifier-4

                    [DELIMITER IN identifier-5]

                    [COUNT IN identifier-6]}...

       [WITH POINTER identifier-7]

       [TALLYING IN identifier-8]

       [ON OVERFLOW imperative-statement-1]

       [NOT ON OVERFLOW imperative-statement-2]

       [END-UNSTRING]
```

Syntax Rules

1. Each literal must be a non-numeric literal. In addition, each literal may be any figurative constant except ALL literal.

2. Identifier-1, identifier-2, identifier-3, and identifier-5 must reference data items described, implicitly or explicitly, as category alphanumeric.

3. Identifier-4 may be described as the category alphabetic, alphanumeric, or numeric (except that the symbol 'P' may not be used in the PICTURE character-string), and must be described, implicitly or explicitly, as USAGE IS DISPLAY.

4. Identifier-6, and identifier-8 must reference integer data items (except that the symbol 'P' may not be used in the PICTURE character-string).

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

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

7. Identifier-1 must not be reference modified.

General Rules

1. All references to identifier-2 and literal-1 apply equally to identifier-3 and literal-2, respectively, and all recursions thereof.

2. The data item referenced by identifier-1 represents the sending area.

3. The data item referenced by identifier-4 represents the data receiving area. The data item referenced by identifier-5 represents the receiving area for delimiters.

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

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

6. The data item referenced by identifier-7 contains a value that indicates a relative character position within the area referenced by identifier-1.

7. The data item referenced by identifier-8 is a counter which is incremented by 1 for each occurrence of the data item referenced by identifier-4 accessed during the UNSTRING operation.

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

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

9. When any examination encounters two contiguous delimiters, the current receiving area is space filled if it is described as alphabetic or alphanumeric, or zero filled if it is described as numeric.

10. Literal-1 or the content of the data item referenced by identifier-2 can contain any character in the computer's character set.

11. Each literal-1 or the data item referenced by identifier-2 represents one delimiter. When a delimiter contains two or more characters, all of the characters must be present in contiguous positions of the sending item, and in the order given to be recognized as a delimiter.

12. When two or more delimiters are specified in the DELIMITED BY phrase, an 'OR' condition exists between them. Each delimiter is compared to the sending field. If a match occurs, the character(s) in the sending field is considered to be a single delimiter. No character(s) in the sending field can be considered a part of more than one delimiter.

    Each delimiter is applied to the sending field in the sequence specified in the UNSTRING statement.

13. When the UNSTRING statement is initiated, the current receiving area is the data item referenced by identifier-4. Data is transferred from the data item referenced by identifier-1 to the data item referenced by identifier-4 according to the following rules:

    a. If the POINTER phrase is specified, the string of characters referenced by identifier-1 is examined beginning with the relative character position indicated by the content of the data item referenced by identifier-7. If the POINTER phrase is not specified, the string of characters is examined beginning with the leftmost character position.

    b. If the DELIMITED BY phrase is specified, the examination proceeds left to right until either a delimiter specified by literal-1 or the value of the data item referenced by identifier-2 is encountered. (See "General Rule" 11). If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current receiving area. However, if the sign of the receiving item is defined as occupying a separate character position, the number of characters examined is one less than the size of the current receiving area.

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

c. The characters thus examined (excluding the delimiting character(s), if any) are treated as an elementary alphanumeric data item, and are moved into the current receiving area according to the rules for the MOVE statement. (See the "MOVE Statement", Chapter 12).

d. If the DELIMITER IN phrase is specified the delimiting character(s) are treated as an elementary alphanumeric data item and are moved into the data item referenced by identifier-5 according to the rules for the MOVE statement (see the "MOVE Statement", Chapter 12). If the delimiting condition is the end of the data item referenced by identifier-1, then the data item referenced by identifier-5 is space filled.

e. If the COUNT IN phrase is specified, a value equal to the number of characters thus examined (excluding the delimiter character(s), if any) is moved into the area referenced by identifier-6 according to the rules for an elementary move.

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

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

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

15. The content of the data item referenced by identifier-7 will be incremented by one for each character examined in the data item referenced by identifier-1. When the execution of an UNSTRING statement with a POINTER phrase is completed, the content of the data item referenced by identifier-7 will contain a value equal to the initial value plus the number of characters examined in the data item referenced by identifier-1.

16. When the execution of an UNSTRING statement with a TALLYING phrase is completed, the content of the data item referenced by identifier-8 contains a value equal to its value at the beginning of the execution of the statement plus a value equal to the number of identifier-4 receiving data items accessed during execution of the statement.

17. Either of the following situations causes an overflow condition:

    a. An UNSTRING is initiated, and the value in the data item referenced by identifier-7 is less than 1 or greater than the size of the data item referenced by identifier-1.

    b. If, during execution of an UNSTRING statement, all receiving areas have been acted upon, and the data item referenced by identifier-1 contains characters that have not been examined.

18. When an overflow condition exists, the UNSTRING operation is terminated, the NOT ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the UNSTRING statement or, if the ON OVERFLOW phrase is specified, to imperative-statement-1. If control is transferred to imperative-statement-1, execution continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the UNSTRING statement.

19. The END-UNSTRING phrase delimits the scope of the UNSTRING statement.

20. If, at the time of execution of an UNSTRING statement, the conditions described in general rule 17 are not encountered, after completion of the transfer of data according to the other general rules, the ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the UNSTRING statement or, if the NOT ON OVERFLOW phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the UNSTRING statement.

21. If identifier-1, identifier-2, or identifier-3, occupies the same storage area as identifier-4, identifier-5, identifier-6, identifier-7, or identifier-8, or if identifier-4, identifier-5, or identifier-6, occupies the same storage area as identifier-7 or identifier-8, or if identifier-7 and identifier-8 occupy the same storage area, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See "Overlapping Operands" in Chapter 10.)

13.13 USE

Description

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.

It is also used to specify Procedure Division statements that are executed just before a report group named in the Report Section of the Data Division is presented.

Format 1

```
                                {EXCEPTION}
USE [GLOBAL] AFTER STANDARD     {          } PROCEDURE
                                {ERROR     }

            {{file-name-1}...}
            { INPUT          }
         ON { OUTPUT         }
            { I-O            }
            { EXTEND         }
```

Format 2

```
USE [GLOBAL] BEFORE REPORTING identifier-2
```

Format 3

```
USE FOR DEBUGGING

            { cd-name-1                       }
            {[ALL REFERENCES OF] identifier-1 }
            {          |--------------------| }
         ON {          | [WITH CONVERSION]  | }...
            {          |--------------------| }
            { procedure-name-1                }
            { file-name-1                     }
            { ALL PROCEDURES                  }
```

Syntax Rules

1. A USE statement, when present, must immediately follow a section header in the declaratives portion of the Procedure Division and must appear in a sentence by itself. The remainder of the section must consist of zero, one or more procedural paragraphs that define the procedures to be used.

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

3. Appearance of file-name-1 in a USE statement must not cause the simultaneous request for execution of more than one USE procedure.

4. In Format 1 the words ERROR and EXCEPTION are synonymous and may be used interchangeably.

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

6. The INPUT, OUTPUT, I-O, and EXTEND phrases may each be specified only once in the declaratives portion of a given Procedure Division.

7. The same file-name can appear in a different specific arrangement of any Format except Format 3. An explanation of the use of Format 3 is given in the appropriate chapter (see the "Debugging Facility", Chapter 16). Appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE procedure.

8. A sort or merge file may only be referenced in a Format 3 USE statement.

9. In Format 2, identifier-2 represents a report group. Identifier-2 must not appear in more than one Format 2 USE statement.

   The GENERATE, INITIATE, or TERMINATE statements must not appear in a paragraph within a USE BEFORE REPORTING procedure. A PERFORM statement in a USE BEFORE REPORTING procedure must not have GENERATE, INITIATE, or TERMINATE statements in its range.

   A USE BEFORE REPORTING procedure must not alter the value of any control data item.

General Rules

All Formats

1. Declarative procedures may be included in any COBOL source program irrespective of whether the program contains or is contained within another program. A declarative is invoked when any of the conditions described in the USE statement which prefaces the declarative occurs while the program is being executed.

   Only a declarative within the separately compiled program that contains the statement which caused the qualifying condition is invoked when any of the conditions described in the USE statement which prefaces the declarative occurs while that separately compiled program is being executed. If no qualifying declarative exists in the separately compiled program, no declarative is executed.

2. Special precedence rules are followed when programs are contained within other programs. In applying these rules, only the first qualifying declarative will be selected for execution. The declarative which is selected for execution must satisfy the rules for execution of that declarative. The order of precedence for selecting a declarative is:

   a. The declarative within the program that contains the statement which caused the qualifying condition.

   b. The declarative in which the GLOBAL phrase is specified and which is within the program directly containing the program which was last examined for a qualifying declarative.

   c. Any declarative selected by applying rule 2b to each more inclusive containing program until rule 2b is applied to the outermost program. If no qualifying declarative is found, none is executed.

3. Within a Declarative procedure, there must not be any reference to any non-declarative procedures.

4. Procedure-names associated with a USE statement may be referenced in a different declarative section or in a non-declarative procedure only with a PERFORM statement.

Rules For Specific Formats

5. When file-name-1, is specified explicitly in a Format 1 USE statement, no other USE statement applies to file-name-1.

6. The procedures associated with a Format 1 USE statement are executed as follows upon the unsuccessful execution of an input-output operation unless an AT END or INVALID KEY phrase takes precedence;

   a. If file-name-1 is specified, the associated procedure is executed when the condition described in the USE statement occurs.

   b. If INPUT is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the input mode or in the process of being opened in the input mode, except those files referenced by file-name-1 in another Format 1 USE statement specifying the same condition.

   c. If OUTPUT is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the output mode or in the process of being opened in the output mode, except those files referenced by file-name-1 in another Format 1 USE statement specifying the same condition.

   d. If I-O is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the I-O mode in the process of being opened in the I-O mode, except those files referenced by file-name-1 in another Format 1 USE statement specifying the same condition.

   e. If EXTEND is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the extend mode or in the process of being opened in the extend mode, except those files referenced by file-name-1 in another Format 1 USE statement specifying the same condition.

7. The procedures designated by a Format 1 USE statement 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 condition, when the INVALID KEY phrase or AT END phrase, respectively, has not been specified in the input-output statement.

8. After execution of the USE procedure, control is transferred to the next executable statement following the input-output statement whose execution caused the exception.

9. 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.

10. In a USE BEFORE REPORTING statement, the designated procedures are executed by the Report Writer Control System just before the named report group is produced (see the "TYPE Clause", Chapter 9).

11. An explanation of the USE FOR DEBUGGING statement is given in the Debugging Facility, Chapter 16.

13.14 WRITE

Description

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

Format 1

```
WRITE record-name-1 [FROM identifier-1]

                             {identifier-2} [LINE ]
                            {{             } [     ]}
      {BEFORE}              {{integer-1    } [LINES]}
     [{      } ADVANCING    {                       }]
      {AFTER }              {{mnemonic-name-1}     }
                            {{                }     }
                             {PAGE            }

          {END-OF-PAGE}
      [AT {            } imperative-statement-1]
          {EOP         }

              {END-OF-PAGE}
      [NOT AT {            } imperative-statement-2]
              {EOP         }

      [END-WRITE]
```

Format 2

```
WRITE record-name-1 [FROM identifier-1]

      [INVALID KEY imperative-statement-1]

      [NOT INVALID KEY imperative-statement-2]

      [END-WRITE]
```

Syntax Rules

1. If identifier-1 is a function-identifier, it must reference an alphanumeric function. When identifier-1 is not a function-identifier, record-name-1 and identifier-1 must not refer to the same storage area.

2. Format 1 must be used for sequential files.

3. Format 2 must be used for mass storage files with other than sequential organization.

4. A WRITE statement must not reference a Report Group Description Entry.

5. Record-name-1 is the name of a logical record in the File Section of the Data Division and may be qualified.

6. Record-name-1 must not be defined within a Sort-Merge File Description entry.

7. The ADVANCING mnemonic-name-1 phrase cannot be specified when writing a record to a file which is associated with a File Description entry containing a LINAGE clause.

8. Identifier-2 must reference an integer data item.

9. Integer-1 may be positive or zero, but must not be negative.

10. When mnemonic-name-1 is specified, the name is associated with a particular feature. The mnemonic name is defined in the SPECIAL-NAMES paragraph of the Environment Division, and must be associated to LNm or CHANNEL-p (see "SPECIAL-NAMES", Chapter 7).

11. The phrases ADVANCING PAGE and END-OF-PAGE must not both be specified in a single WRITE statement.

12. If the END-OF-PAGE or the NOT END-OF-PAGE is specified, the LINAGE clause must be specified in the File Description entry for the associated file.

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

14. When Format 2 is used, the INVALID KEY phrase must be specified if an applicable USE AFTER STANDARD EXCEPTION procedure is not specified for the associated file-name.

|15. The ADVANCING phrase cannot be specified when writing a record to a file whose SELECT clause contains the WITH ASA phrase.|

General Rules

All Files

1. The file referenced by the file-name associated with record-name-1 must be open in the Output, I-O, or Extend mode at the time of the execution of this statement (see the "OPEN Statement", Chapter 12).

2. The logical record released by the successful execution of the WRITE statement is no longer available in the record area unless the file-name associated with record-name-1 is specified in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as the file associated with record-name-1.

3. The result of the execution of a WRITE statement with the FROM phrase is equivalent to the execution of the following statements in the order specified:

   a. The statement: MOVE identifier-1 to record-name-1 according to the rules specified for the MOVE statement.

   b. The same WRITE statement without the FROM phrase.

4. After the 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-1 is not available except as specified by the SAME RECORD AREA clause.

5. The file position indicator is not affected by the execution of a WRITE statement.

6. The execution of the WRITE statement causes the value of the I-O status of the file-name associated with record-name-1 to be updated (See "I-O Status", Chapter 7).

7. The execution of the WRITE statement releases a logical record to the operating system.

8. The number of character positions in the record referenced by record-name-1 must not be larger than the largest or smaller than the smallest number of character positions allowed by the RECORD IS VARYING clause associated with the file-name associated with record-name-1. In either of these cases the execution of the WRITE statement is unsuccessful, the WRITE operation does not take place, the content of the record area is unaffected and the I-O status of the file associated with record-name-1 is set to a value indicating the cause of the condition.

9. Transfer of control following the successful or unsuccessful execution of the WRITE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the WRITE statement. (See the "Invalid Key Condition" in Chapter 10.)

10. The END-WRITE phrase delimits the scope of the WRITE statement. A description of the function of the END-WRITE phrase is given in the appropriate paragraph. (See "Scope of Statements" in Chapter 10.)

Sequential Files

11. The successor relationship of a sequential file is established by the order of execution of WRITE statements when the file is created. The relationship does not change except when records are added to the end of a file.

12. When a sequential file is open in the extend mode, the execution of the WRITE statement will add records to the end of the file as though the file were open in the output mode. If there are records in the file, the first record written after the execution of the OPEN statement with the EXTEND phrase is the successor of the last record in the file.

13. 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 actions take place:

    a. The value of the I-O status of the file-name associated with record-name-1 is set to a value indicating a boundary violation (See "I-O Status", Chapter 7).

    b. If a USE AFTER STANDARD EXCEPTION declarative is explicitly or implicitly specified for the file-name associated with record-name-1, that declarative procedure will then be executed.

    c. If a USE AFTER STANDARD EXCEPTION declarative is not explicitly or implicitly specified for the file-name associated with record-name-1, the result is undefined.

14. If the end of a reel/unit is recognized and the externally defined boundaries of the file have not been exceeded, the following operations are executed:

    a. The standard ending reel/unit label procedure.

    b. A reel/unit swap. The current volume pointer is updated to point to the next reel/unit existing for the file.

    c. The standard beginning reel/unit label procedure.

15. Both the ADVANCING phrase and the END-OF-PAGE phrase allow control of the vertical positioning of each line on a representation of a printed page.

    If the ADVANCING phrase is not used, automatic advancing will occur as if the user had specified AFTER ADVANCING 1 LINE |, unless the SELECT clause given for the file-name contains the WITH ASA phrase. In that case, the first character of the record is interpreted by the Operating System as an advancing information and is not presented on the printed line. If the WITH ASA phrase, appears in the SELECT clause, advancing is provided as follows, according to the first characters of the record.

(S) AFTER ADVANCING 1 LINE
0 (G) AFTER ADVANCING 2 LINES
- (H) AFTER ADVANCING 3 LINES
+ (I) AFTER ADVANCING O LINES
1 (J) AFTER ADVANCING mnemonic-name-1 (LN-m,m=head of page)
2 (K) AFTER ADVANCING mnemonic-name-1 (CHANNEL-2)
3 (L) AFTER ADVANCING mnemonic-name-1 (CHANNEL-3)
4 (M) AFTER ADVANCING mnemonic-name-1 (CHANNEL-4)
5 (N) AFTER ADVANCING mnemonic-name-1 (CHANNEL-5)
6 (O) AFTER ADVANCING mnemonic-name-1 (CHANNEL-6)
7 (P) AFTER ADVANCING mnemonic-name-1 (CHANNEL-7)
8 (Q) AFTER ADVANCING mnemonic-name-1 (CHANNEL-8)
9 (R) AFTER ADVANCING mnemonic-name-1 (CHANNEL-9)
A (D) AFTER ADVANCING mnemonic-name-1 (CHANNEL-10)
B (E) AFTER ADVANCING mnemonic-name-1 (CHANNEL-11)
C (F) AFTER ADVANCING mnemonic-name-1 (CHANNEL-12)|

|Both the first character and the following one enclosed between parentheses have the same effect, except when a printer PRU 0711 is used. In that case the character between parentheses must be used when CMC7 printing is requested.|

If the ADVANCING phrase is used, advancing is provided as follows:

a. If integer-1 or the value of the data item referenced by identifier-2 is positive, the representation of the printed page is advanced the number of lines equal to that value.

b. If the value of the data item referenced by identifier-2 is negative, the results are undefined.

c. If integer-1 or the value of the data item referenced by identifier-2 is zero, no repositioning of the representation of the printed page is performed.

|d. If mnemonic name is specified, the representation of the printed page is advanced according to the following rules:

(i) If mnemonic-name-1 is associated to LN-m, the printer page is advanced to the line whose absolute number in the page is "m" as specified in the related LN-m.

(ii) If mnemonic-name-1 is associated to CHANNEL-p, the printer page is advanced to a position governed by the "p"th, channel of the vertical-format paper tape loop, "p" being as specified in the related CHANNEL-p.|

e. If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced according to RULES a, b, c, |and d| above.|

f. If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced according to Rules a, b, c, |and d| above.

g. If PAGE is specified and the LINAGE clause is specified in the associated File Description entry, 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. The repositioning is to the first line that can be written on the next logical page as specified in the LINAGE clause.

h. If PAGE is specified and the LINAGE clause is not specified in the associated File Description entry, the record is presented on the physical page before or after (depending on the phrase used) the device is repositioned to the next physical page. The repositioning to the next physical page is accomplished when the physical page is full. If physical page has no meaning in conjunction with a specific device, advancing will act as if the user had specified BEFORE or AFTER (depending on the phrase used) ADVANCING 1 LINE.

16. If the logical end of the representation of the printed page is reached during the execution of a Format 1 WRITE statement with the END-OF-PAGE phrase, imperative-statement-1 specified in the END-OF-PAGE phrase is executed. The logical end is specified in the LINAGE clause associated with record-name-1.

17. An end-of-page condition occurs when the execution of a given WRITE statement with the END-OF-PAGE phrase causes printing or spacing within the footing area of a page body. This occurs when the execution of such a WRITE statement causes the LINAGE-COUNTER to equal or exceed the value specified by integer-2 or the data item referenced by data-name-2 of the LINAGE clause, if specified. In this case, the WRITE statement is executed and then imperative-statement-1 in the END-OF-PAGE phrase is executed.

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

    This occurs when a WRITE statement, if executed, would cause the LINAGE-COUNTER to exceed the value specified by integer-1 or the data item referenced by data-name-1 of the LINAGE clause. In this case, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the first line that can be written on the next logical page as specified in the LINAGE clause. Imperative-statement-1 in the END-OF-PAGE phrase, if specified, is executed after the record is written and the device has been repositioned.

    A page overflow condition occurs when the execution of a given WRITE statement would cause LINAGE-COUNTER to simultaneously exceed the value of both integer-2 or the data item referenced by data-name-2 of the LINAGE clause and integer-1 or the data item referenced by data-name-1 of the LINAGE clause.

    If integer-2 or data-name-2 of the LINAGE clause is not specified, no end-of-page condition distinct from the page overflow condition is detected. In this case, the end-of-page condition and page overflow condition occur simultaneously.

Relative Files

18. When a relative 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 causes a record to be released to the Operating System. The first record will have a relative record number of one (1), and subsequent records released will have relative record numbers of 2, 3, 4,... If the RELATIVE KEY phrase is specified for the file-name associated with record-name-1, the relative record number of the record being released is moved into the relative key data item during execution of the WRITE statement according to the rules for the MOVE 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 initialized by the program with the relative record number to be associated with the record in the record area. That record is then released to the Operating System by execution of the WRITE statement.

19. When a relative file is open in the extend mode, records are inserted into the file. The first record released has a relative record number one greater than the highest relative record number existing in the file. Subsequent records released have consecutively higher relative record numbers. If the RELATIVE KEY phrase is specified for the file-name associated with record-name-1, the relative record number of the record being released is moved into the relative key data item by the Operating System during execution of the WRITE statement according to the rules for the MOVE statement.

20. When a relative 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 initialized by the program with the relative record number to be associated with the record in the record area. Execution of a Format 2 WRITE statement then causes the content of the record area to be released to the Operating System.

21. 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.

22. When the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file, the result will be unpredictable.

23. When the invalid key condition is recognized, the execution of the WRITE statement is unsuccessful, the content of the record area is unaffected, and the I-O status of the file-name associated with record-name-1 is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules for an invalid key condition (see "I-O STATUS", Chapter 7, and the "Invalid Key Condition" in Chapter 10).

Indexed Files

24. Execution of a Format 2 WRITE statement causes the content of the record area to be released. The Operating System utilizes the contents of the Record Keys in such a way that subsequent access of the record may be made based upon any of those specified Record Keys.

25. The value of the Prime Record Key must be unique within the records in the file.

26. The data item specified as the Prime Record Key must be set by the program to the desired value prior to the execution of the WRITE statement.

27. If the file is open in the sequential access mode, records must be released to the Operating System in ascending order of Prime Record Key values according to the collating sequence of the file. When the file is open in the extend mode, the first record released to the Operating System must have a Prime Record Key whose value is greater than the highest Prime Record Key value existing in the file.

28. If the file is open in the random or dynamic access mode, records may be released to the Operating System in any program-specified order.

29. When the ALTERNATE RECORD KEY clause is specified in the file control entry for an indexed file, the value of the Alternate Record Key may be non-unique only if the DUPLICATES phrase is specified for that data item. In this case the Operating System provides storage of records such that when records are accessed sequentially, the order of retrieval of those records is the order in which they are released to the Operating System.

30. The invalid key condition exists under the following circumstances:

    a. When the file is open in the sequential access mode, and the file also is open in the output or extend mode, and the value of the Prime Record Key is not greater than the value of the Prime Record Key of the previous record, or

    b. When the file is open in the output or I-O mode, and the value of the Prime Record Key is equal to the value of a Prime Record Key of a record already existing in the file, or

    c. When the file is open in the output, extend or I-O mode, and the value of an Alternate Record Key for which duplicates are not allowed equals the value of the corresponding data item of a record already existing in the file, or

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

31. When the invalid key condition is recognized the execution of the WRITE statement is unsuccessful, the content of the record area is unaffected and the I-O status of the file-name associated with record-name-1 is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules for an invalid key condition (see "I-O STATUS", Chapter 7, and the "Invalid Key Condition" in Chapter 10).

14. Segmentation

14.1 GENERAL DESCRIPTION

COBOL segmentation is a facility that provides a means by which the user may communicate with the compiler to specify object program overlay requirements.

The segmentation is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

14.1.1 Scope

COBOL segmentation deals only with segmentation of procedures. As such, only the Procedure Division and the Environment Division are considered in determining segmentation requirements for an object program.

14.1.2 Organization

14.1.2.1 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. Each section must then be specified as belonging to either the non-overlayable or the overlayable portion of the program.

The use of segmentation only affects the physical management of the object program during execution. Segmentation neither imposes any syntactic restrictions nor implies any semantic differences over the same program written without segmentation.

14.1.2.2 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 two types of segments: fixed permanent segments and fixed overlayable segments.

A fixed permanent segment is a segment in the fixed portion which cannot be overlaid by any other part of the program. A fixed overlayable segment is a segment in the fixed portion which, although logically treated as if it were always in memory, can be overlaid by another segment to optimize memory utilization. Variation of the number of fixed permanent segments in the fixed portion can be accomplished by using a special facility called the SEGMENT-LIMIT clause (see "SEGMENT-LIMIT Clause", this chapter). Such a segment, if called for by the program, is always made available in its last used state.

14.1.2.3 Independent Segments

An independent segment is defined as part of the object program which can overlay, and can be overlaid by, either a fixed overlayable segment or another independent segment. An independent segment is in its initial state whenever control is transferred (either implicitly or explicitly) to that segment for the first time during the execution of a program. On subsequent transfers of control to the segment, an independent segment is also in its initial state when:

1. Control is transferred to that segment as a result of the implicit transfer of control between consecutive statements from a segment with a different segment-number.

2. Control is transferred to that segment as the result of the implicit transfer of control between a SORT or MERGE statement, in a segment with a different segment-number, and an associated input or output procedure in that independent segment.

3. Control is transferred explicitly to that segment from a segment with a different segment-number (with the exception noted in paragraph 2 below).

On subsequent transfer of control to the segment, an independent segment is in its last-used state when:

1. Control is transferred implicitly to that segment from a segment with a different segment-number (except as noted in paragraphs 1 and 2 above).

2. Control is transferred explicitly to that segment as the result of the execution of an EXIT PROGRAM statement.

See "Explicit and Implicit Transfers of Control", Chapter 3.

14.1.3 Segment Classification

Sections which are to be segmented are classified, using a system of segment-numbers (see "Segment Numbers", this chapter) 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 either to one of the overlayable fixed segments or 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.

14.1.4 Segmentation Control

The logical sequence of the program is the same as the physical sequence except for specific transfers of control. If any reordering of the object program is required to handle the flow from segment to segment, according to the rules given for Segment Numbers, control transfers are provided to maintain the logical flow specified in the source program. All controls necessary for a segment to operate whenever the segment is used are also provided. 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.

14.2 STRUCTURE OF PROGRAM SEGMENTS

14.2.1 Segment Numbers

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

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.
2. Segments with segment-number 0 through 49 belong to the fixed portion of the object program.
3. Segments with segment-number 50 through 99 are independent segments.

14.2.2 SEGMENT-LIMIT Clause

Ideally, all program segments having segment-numbers ranging from 0 through 49 should be specified as permanent segments. However, when the user wishes to control the size of 'truly' permanent segments it may become necessary to decrease the number of permanent segments. The SEGMENT-LIMIT feature provides the user with a means by which he can reduce the number of permanent segments in his program, while still retaining the logical properties of fixed portion segments (segment-number 0 through 49). The SEGMENT-LIMIT clause appears in the OBJECT-COMPUTER paragraph.

Format

```
[SEGMENT-LIMIT IS segment-number]
```

Syntax Rule

Segment-number must be an integer ranging in value from 1 to 49.

General Rules

1. When the SEGMENT-LIMIT clause is specified, only those segments having segment-numbers from 0 up to, but not including, the segment-number designated as the segment-limit, are considered as permanent segments of the object program.

2. Those segments having segment-numbers from the segment-limit through 49 are considered as overlayable fixed segments.

3. When the SEGMENT-LIMIT clause is omitted, all segments having segment-numbers from 0 through 49 are considered as permanent segments of the object program.

14.3 RESTRICTIONS ON PROGRAM FLOW

When segmentation is used, the following restrictions are placed on the ALTER, PERFORM, MERGE, and SORT statements.

14.3.1 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.

All other uses of the ALTER statement are valid and are performed even if the GO TO to which the ALTER refers is in a fixed overlayable segment.

14.3.2 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.

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

2. Sections and/or paragraphs 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:

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

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

14.3.3 The MERGE Statement

If the MERGE statement appears in a section that is not in an independent segment, then any output procedure referenced by that MERGE statement must appear:

1. Totally within non-independent segments, or

2. Wholly contained in a single independent segment.

If a MERGE statement appears in an independent segment, then any output procedure referenced by that MERGE statement must be contained:

1. Totally within non-independent segments, or

2. Wholly within the same independent segment as that MERGE statement.

14.3.4 The SORT Statement

If a SORT statement appears in a section that is not an independent segment, then any input procedures or output procedures referenced by that SORT statement must appear:

1. Totally within non-independent segments, or

2. Wholly contained in a single independent segment.

If a SORT statement appears in an independent segment, then any input procedures or output procedures referenced by that SORT statement must be contained:

1. Totally within non-independent segments, or

2. Wholly within the same independent segment as that SORT segment.

15. COBOL Source Text Manipulation Facilities

15.1 INTRODUCTION

The COBOL library facilities are the COPY statement and the REPLACE statement. Each of these facilities can function either independently of the other or in conjunction with the other to provide an extensive capability to insert and replace source program text as part of the compilation of the source program.

COBOL libraries contain library texts that are available to the compiler at compile time. The effect of the interpretation of the COPY statement is to generate, from a library text, text which is treated by the compiler as part of the source program.

Similarly, COBOL source programs can be written in a programmer defined notation which, at compile time, can be expanded into syntactically correct phrases, clauses, and statements. The effect of the interpretation of the REPLACE statement is to substitute new text for text appearing in the source program and have the substituted text treated by the compiler as part of the source program.

15.2 COPY

Description

The COPY statement incorporates text into a COBOL source program.

Format

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

     [ REPLACING

          {== pseudo-text-1 ==}    {== pseudo-text-2 ==}
          {identifier-1       }    {identifier-2       }
        {   {                 } BY {                   }}
        {   {literal-1        }    {literal-2          }}
        {   {word-1           }    {word-2             }}... ]
        {                                                 }
        { |-----------------------------------|           }
        { | {LEADING }             {literal-4} |           }
        { | {        } literal-3 BY {SPACE    } |           }
          | {TRAILING}             {SPACES   } |
          |-----------------------------------|
```

Syntax Rules

1. If more than one COBOL library is available during compilation, text-name must be qualified by the library-name identifying the COBOL library in which the text associated with text-name resides.

   Within one COBOL library, each text-name must be unique.

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

3. Pseudo-text-1 must contain one or more text words.

4. Pseudo-text-2 may contain zero, one or more text words.

5. Character-strings within pseudo-text-1 and pseudo-text-2 may be continued. (See "Continuation of Lines", Chapter 4.)

6. Word-1 or word-2 may be any single COBOL word, except 'COPY'.

|7. Literal-3 and literal-4 must be non-numeric literals.|

8. A COPY statement may occur in the source program anywhere a character-string or a separator, other than the quotation mark, may occur except that a COPY statement must not occur within a COPY statement |, nor in a REPLACE statement of the Substitution Section of the Control Division.|

9. A text word within pseudo-text and within library text must not exceed 322 characters in length.

10. Pseudo-text-1 must not consist entirely of a separator comma or a separator semi-colon.

11. If the word COPY appears in a comment-entry or in the place where a comment-entry may appear, it is considered part of the comment-entry.

General Rules

1. The compilation of a source program containing COPY statements is logically equivalent to processing all COPY statements prior to the processing of the resultant source program.

2. The effect of processing a COPY statement is that the library text associated with text-name is copied into the source program, logically replacing the entire COPY statement, beginning with the reserved word COPY and ending with the punctuation character period, inclusive,

3. If the REPLACING phrase is not specified, the library text is copied unchanged.

   If the REPLACING phrase is specified, the library text is copied and each properly matched occurrence of pseudo-text-1, identifier-1, word-1, literal-1 |and leading or trailing literal-3 in words| in the library text is replaced by the corresponding pseudo-text-2, identifier-2, word-2, literal-2, |literal-4 or deleted.|

4. For purposes of matching, identifier-1, word-1, and literal-1 are treated as pseudo-text containing only identifier-1, word-1, or literal-1, respectively.

5. The comparison operation to determine text replacement occurs in the following manner:

   a. The leftmost library text word which is not a separator comma or a separator semi-colon, is the first text word used for comparison. Any text word or space preceding this text word is copied into the source program. Starting with the first text word for comparison and the first pseudo-text-1, identifier-1, word-1, literal-1 |or literal-3| that was specified in the REPLACING phrase, the entire REPLACING phrase operand that precedes the reserved word BY is compared to an equivalent number of contiguous library text words, |or in the case of literal-3 to the equivalent number of LEADING or TRAILING characters of the leftmost text word when it is a word.|

b. Pseudo-text-1, identifier-1, word-1, or literal-1 match the library text if, and only if, the ordered sequence of text words that forms pseudo-text-1, identifier-1, word-1, or literal-1 is equal, character for character, to the ordered sequence of library text words. For purposes of matching, each occurrence of a separator comma or semi-colon in pseudo-text-1 or in the library text is considered to be a single space. Each sequence of one or more space separators is considered to be a single space.

|c. Literal-3 matches the source program text if, and only if, the leftmost text word commences (LEADING) or finishes (TRAILING) with the same sequence of characters as that forms literal-3.|

d. If no match occurs, the comparison is repeated with each next successive pseudo-text-1, identifier-1, word-1, literal-1, |or literal-3|, if any, in the REPLACING phrase until either a match is found or there is no next successive REPLACING operand.

e. When all the REPLACING phrase operands have been compared and no match has occurred, the leftmost library text word is copied into the source program. The next successive library text word is then considered as the leftmost library text word, and the comparison cycle starts again with the first pseudo-text-1, identifier-1, word-1, literal-1, |or literal-3,| specified in the REPLACING phrase.

f. Whenever a match occurs between pseudo-text-1, identifier-1, word-1, or literal-1 and the library text, the corresponding pseudo-text-2, identifier-2, word-2 or literal-2 is placed into the source program. |Whenever a match occurs between literal-3 and a text word, the matching leading or trailing characters of the word are either replaced by the characters that form literal-4, or if the SPACE or SPACES phrase is used, are deleted; the replacement is done only if it results in a legal word.| The library text word immediately following the rightmost text word that participated in the match is then considered as the leftmost library text word. The comparison cycle starts again with the first pseudo-text-1, identifier-1, word-1, literal-1, |or literal-3,| specified in the REPLACING phrase.

g. The comparison operation continues until the rightmost text word in the library text has either participated in a match or been considered as a leftmost library text word and participated in a complete comparison cycle.

6. Comment lines occurring in the library text and pseudo-text-1 are ignored for purpose of matching; and the sequence of text words in the library text, if any, and in pseudo-text-1 is determined by the rules for reference format. (See "Reference Format Representation", Chapter 17). Comment lines or blank lines appearing in pseudo-text-2 are copied into the resultant program unchanged whenever pseudo-text-2 is placed into the source program as a result of text replacement. Comment lines or blank lines appearing in library text are copied into the resultant source program unchanged with the following exception: comment line or blank line in library text is not copied if that comment line or blank line appears within the sequence of text words that match pseudo-text-1.

7. Debugging lines are permitted within library text and pseudo-text. Text words within a debugging line participate in the matching rules as if the 'D' did not appear in the Indicator Area. A debugging line is specified within pseudo-text if the debugging line begins in the source program after the opening pseudo-text-delimiter but before the matching closing pseudo-text-delimiter.

8. The syntactic correctness of the library text cannot be independently determined. Except for COPY and REPLACE statements, the syntactic correctness of the entire COBOL source program cannot be determined until all COPY and REPLACE statements have been completely processed.

9. Each text word copied from the library but not replaced is copied so as to start in the same area of the line in the resultant program as it begins in the line within the library. However, if a text word copied from the library begins in area A but follows another text word, which also begins in area A of the same line, and if replacement of a preceding text word in the line by replacement text of greater length occurs, the following text word begins in area B if it cannot begin in area A. Each text word in pseudo-text-2 that is to be placed in the resultant program begins in the same area of the resultant program as it appears in pseudo-text-2. Each identifier-2, literal-2, and word-2 that is to be placed into the resultant program begins in the same area of the resultant program as the leftmost library text word that participated in the match would appear if it had not been replaced.

   Library text must conform to the rules for COBOL Reference Format.

   If additional lines are introduced into the source program as a result of a COPY statement, each text word introduced appears on a debugging line if the COPY statement begins on a debugging line or if the text word being introduced appears on a debugging line in library text. When a text word specified in the BY phrase is introduced, it appears on a debugging line if the first library text word being replaced is specified on a debugging line. Except in the preceding cases, only those text words that are specified on debugging lines where the debugging line is within pseudo-text-2 appear on debugging lines in the resultant program. If any literal specified as literal-2 or within pseudo-text-2 or library text is of too great length to be accommodated on a single line without continuation to another line in the resultant program and the literal is not being placed on a debugging line, additional continuation lines are introduced which contain the remainder of the literal. If replacement requires that the continued literal be continued on a debugging line, the program is in error.

10. For purposes of compilation, text words after replacement are placed in the source program according to the rules for Reference Format, (see Chapter 17).When copying text words of pseudo-text-2 into the source program, additional spaces may be introduced only between text words where there already exists a space (including the assumed space between source lines.

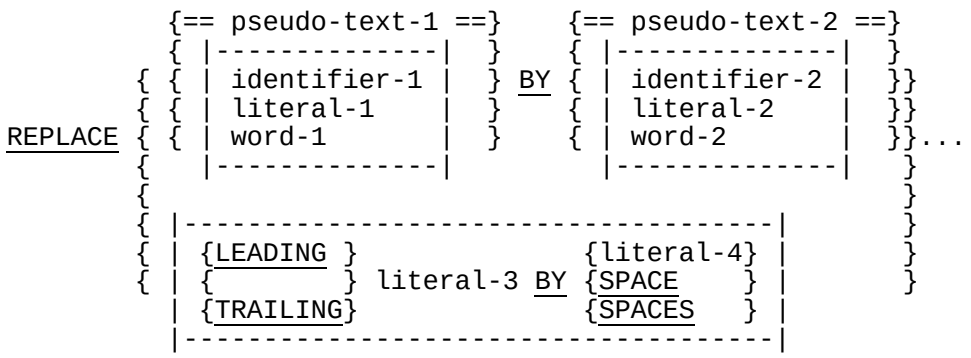
11. If additional lines are introduced into the source program as a result of the processing of COPY statements, the indicator area of the introduced line contains the same character as the line on which the text being replaced begins, unless that line contains an hyphen, in which case the introduced line contains a space. In the case where a literal is continued onto an introduced line which is not a debugging line, a hyphen is placed in the indicator area.

15.3 REPLACE

Description

The REPLACE statement is used to replace source program text.

Format 1



Format 2

```
REPLACE OFF
```

Syntax Rules

1. A REPLACE statement may occur anywhere in the source program where a character-string may occur |except in the Substitution Section of the Control Division.| It must be preceded by a separator period except when it is the first statement in a separately compiled program.

2. A REPLACE statement must be terminated by a separator period.

3. Pseudo-text-1 must contain one or more text words.

4. Pseudo-text-2 must contain zero, one or more text words.

5. Character-strings within pseudo-text-1 and pseudo-text-2 may be continued. (See "Reference Format", Chapter 17).

6. A text word within pseudo-text must not exceed 322 characters in length.

7. Pseudo-text-1 must not consist entirely of a separator comma or a separator semi-colon.

8. If the word REPLACE appears in a comment-entry or in the place where a comment-entry may appear, it is considered part of the comment-entry.

General Rules

1. The format 1 REPLACE statement specifies the text of the source program to be replaced by the corresponding text. Each matched occurrence of pseudo-text-1 in the source program is replaced by the corresponding pseudo-text-2.

2. The format 2 REPLACE statement specifies that any text replacement currently in effect is discontinued.

3. A given occurrence of the REPLACE statement is in effect from the point at which it is specified until the next occurrence of the statement or the end of the separately compiled program, respectively.

4. Any REPLACE statements contained in a source program are processed after the processing of any COPY statements contained in a source program.

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

|6. For purposes of matching, identifier-1, word-1, and literal-1 are treated as pseudo-text containing only identifier-1, word-1, or literal-1, respectively.|

7. The comparison operation to determine text replacement occurs in the following manner:

   a. Starting with the leftmost source program text word and the first pseudo-text-1 |or literal-3,| pseudo-text-1 is compared to an equivalent number of contiguous source program text words, |or in the case of literal-3 to the equivalent number of LEADING or TRAILING characters of the leftmost text word when it is a word.|

   b. Pseudo-text-1 matches the source program text if and only if the ordered sequence of text words that forms pseudo-text-1 is equal, character for character, to the ordered sequence of source program text words. For purposes of matching, each occurrence of a separator comma, semi-colon, or space in pseudo-text-1 or in the source program text is considered to be a single space. Each sequence of one or more space separators is considered to be a single space.

   |c. Literal-3 matches the source program text if, and only if, the leftmost text word commences (LEADING) or finishes (TRAILING) with the same sequence of characters as that the forms literal-3.|

   d. If no match occurs, the comparison is repeated with each next successive occurrence of pseudo-text-1 |or literal-3,| a match is found or there is no next successive occurrence of pseudo-text-1 |or literal-3.|

   e. When all occurrences of pseudo-text-1 |or literal-3| have been compared and no match has occurred, the next successive source program text word is then considered as the leftmost source program text word, and the comparison cycle starts again with the first occurrence of pseudo-text-1 |or literal-3|.

f. Whenever a match occurs between pseudo-text-1 and the source program text, the corresponding pseudo-text-2, |identifier-2, literal-2 or word-2| replaces the matched text in the source program. |Whenever a match occurs between literal-3 and a text.word, the matching leading or trailing characters of the word are either replaced by the characters that form literal-4, or if the SPACE or SPACES phrase is used, are deleted; the replacement is done only if it results in a legal word.| The source program text word immediately following the rightmost text word that participated in the match is then considered as the leftmost source program text word. The comparison cycle starts again with the first occurrence of pseudo-text-1 |or literal-3.|

g. The comparison operation continues until the rightmost text word in the source program text which is within the scope of the REPLACE statement has either participated in a match or been considered as a leftmost source program text word and participated in a complete comparison cycle.

8. Comment lines or blank lines occurring in the source program text and in pseudo-text-1 are ignored for purposes of matching; and the sequence of text words in the source program text and in pseudo-text-1 is determined by the rules for reference format. (See "Reference Format Representation", Chapter 17). Comment lines or blank lines in pseudo-text-2 are placed into the resultant program unchanged whenever pseudo-text-2 is placed into the source program as a result of text replacement. A comment line or blank line in source program text is not replaced if that comment line or blank line appears within the sequence of text words that match pseudo-text-1.

9. Debugging lines are permitted in pseudo-text. Text words within a debugging line participate in the matching rules as if the 'D' did not appear in the indicator area.

10. Except for COPY and REPLACE statements, the syntactic correctness of the source program text cannot be determined until after all COPY and REPLACE statements have been completely processed.

11. Text words inserted into the source program as a result of processing a REPLACE statement are placed in the source program according to the rules for reference format. (See "Reference Format", Chapter 17). When inserting text words of pseudo-text-2 into the source program, additional spaces may be introduced only between text words where there already exists a space (including the assumed space between source lines).

12. If additional lines are introduced into the source program as a result of the processing of REPLACE statements, the indicator area of the introduced lines contains the same character as the line on which the text being replaced begins, unless that line contains a hyphen, in which case the introduced line contains a space.

    If any literal within pseudo-text-2 is of a length too great to be accommodated on a single line without continuation to another line in the resultant program and the literal is not being placed on a debugging line, additional continuation lines are introduced which contain the remainder of the literal. If replacement requires the continued literal to be continued on a debugging line, the program is in error.

16. Debugging Facility

16.1 INTRODUCTION

COBOL debugging is a facility that provides a means by which the user can describe his debugging algorithm including the conditions under which data or procedure items are to be monitored during the execution of the object program.

The decisions of what to monitor and what information to display on the output device are explicitly in the domain of the user. The COBOL debug facility simply provides a convenient access to pertinent information.

Except for the debugging lines and the WITH DEBUGGING MODE concerning these lines, the debugging facility is an obsolete element in Standard COBOL because it is to be deleted from the next version or Standard COBOL.

16.2 CONCEPTS

The features of the COBOL language that support the debugging facility are:

1. A compile-time switch WITH DEBUGGING MODE |, which can be superseded by the DEBUGMD and NDEGUGMD parameters of the CBL JCL statement.|
2. An object-time switch.
3. A USE FOR DEBUGGING statement.
4. A special register DEBUG-ITEM.
5. Debugging Lines.

16.3 A COMPILE-TIME SWITCH

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

When the WITH DEBUGGING MODE clause is specified in a program and the NDEBUGMD parameter is not specified in the $COBOL JCL statement, or if the DEBUGMD parameter is specified,| all debugging sections and all debugging lines are compiled as specified in this section of the document.

When the WITH DEBUGGING MODE clause is not specified and the DEBUGMD parameter is not specified in the $COBOL JCL statement, or if the NDEBUGMD parameter is specified,| all debugging lines and all debugging sections are compiled as if they were comment lines.

16.4 AN OBJECT-TIME SWITCH

An object time switch dynamically activates the debugging code inserted by the compiler. This switch is operated by using the DEBUG parameter of the $STEP JCL statement. If the switch is 'on', (i.e. if the DEBUG option is present), all the effects of the debugging language written in the source program are permitted. If the switch is 'off', all the effects for the USE FOR DEBUGGING statement are inhibited.

Unless the Program Checkout Facility is used, the object time switch has no effect on the execution of the object program if the WITH DEBUGGING MODE clause is not specified in the source program |and the DEBUGMD parameter is not specified in the $COBOL JCL statement or if the NDEBUGMD parameter is specified.|

16.5 THE USE FOR DEBUGGING STATEMENT

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

Format

```
section-name SECTION [segment-number]

                     { cd-name-1                        }
                     {[ALL REFERENCES OF] identifier-1}
                     {           |--------------------| }
USE FOR DEBUGGING ON {           | [WITH CONVERSION] | }...
                     {           |--------------------| }
                     { procedure-name-1                 }
                     { file-name-1                      }
                     { 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 identifier, cd-name, file-name, or procedure-name may appear in only one USE FOR DEBUGGING statement and may appear only once in that statement.

7. The ALL PROCEDURES phrase can appear only once in a program.

8. When the ALL PROCEDURES phrase is specified, procedure-name-1 must not be specified in any USE FOR DEBUGGING statement.

9. Identifier-1 must not reference any data item defined in the Report Section except sum counters.

10. If the data description entry of the data item referenced by identifier-1 contains an OCCURS clause or is subordinate to a data description entry that contains an OCCURS clause, identifier-1 must be specified without the subscripting or indexing normally required.

11. References to the special register DEBUG-ITEM are restricted to references from within a debugging section.

|12. Identifier-1 must reference an elementary numeric data item if the WITH CONVERSION phrase is specified.|

13. Identifier-1 must not be reference modified.

General Rules

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

2. When file-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:

    a. After the execution of any OPEN or CLOSE statement that references file-name-1, and

    b. After the execution of any READ statement (after any other specified USE procedure) not resulting in the execution of an associated AT END or INVALID KEY imperative statement, and

    c. After the execution of any DELETE or START statement that references file-name-1.

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 4 to occur for every procedure-name in the program, except those appearing within a debugging section.

5. When the ALL REFERENCES OF identifier-1 phrase is specified, that debugging section is executed for every statement that explicitly references identifier-1 at each of the following times:

    a. In the case of a WRITE or REWRITE statement immediately before the execution of that WRITE or REWRITE statement and after the execution of any implicit move resulting from the presence of the FROM phrase.

    b. In the case of a GO TO statement with a DEPENDING ON phrase, immediately before control is transferred and prior to the execution of any debugging section associated with the procedure-name to which control is to be transferred.

c. In the case of a PERFORM statement in which a VARYING, AFTER, or UNTIL phrase references identifier-1, immediately after each initialization, modification or evaluation of the contents of the data item referenced by identifier-1.

d. In the case of any other COBOL statement, immediately after execution of that statement.

If identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

6. When identifier-1 is specified without the ALL REFERENCES OF phrase, that debugging section is executed at each of the following times:

   a. In the case of a WRITE or REWRITE statement that explicitly references identifier-1, immediately before the execution of that WRITE or REWRITE statement and after the execution of any implicit move resulting from the presence of the FROM phrase.

   b. In the case of a PERFORM statement in which a VARYING, AFTER or UNTIL phrase references identifier-1, immediately after each initialization, modification or evaluation of the contents of the data item referenced by identifier-1.

   c. Immediately after the execution of any other COBOL statement that explicitly references and causes the contents of the data item referenced by identifier-1 to be changed.

   If identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

7. 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 causes 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.

8. When cd-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:

   a. After the execution of any ENABLE, DISABLE, and SEND statement that references cd-name-1,

   b. After the execution of a RECEIVE statement referencing cd-name-1 that does not result in the execution of the NO DATA imperative-statement, and

   c. After the execution of an ACCEPT MESSAGE COUNT statement that references cd-name-1.

9. A reference to identifier-1, cd-name-1, file-name-1 or procedure-name-1 as a qualifier does not constitute reference to that item for the debugging described in the general rules above.

10. 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     VALUE SPACE.
   02 DEBUG-SUB-1      PICTURE IS S9999
                       SIGN IS LEADING SEPARATE CHARACTER.
   02 FILLER           PICTURE IS X     VALUE SPACE.
   02 DEBUG-SUB-2      PICTURE IS S9999
                       SIGN IS LEADING SEPARATE CHARACTER.
   02 FILLER           PICTURE IS X     VALUE SPACE.
   02 DEBUG-SUB-3      PICTURE IS S9999
                       SIGN IS LEADING SEPARATE CHARACTER.
   02 FILLER           PICTURE IS X     VALUE SPACE.
   02 DEBUG-CONTENTS   PICTURE IS X(n).
```

11. Prior to each execution of a debugging section, the content of the data item referenced by DEBUG-ITEM is 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. |However, if WITH CONVERSION is specified, DEBUG-CONTENTS contain the contents of a USAGE DISPLAY data item of the same PICTURE as identifier-1 except that it is always signed (with SIGN IS LEADING SEPARATE); if the PICTURE clause is not specified, the data item is however converted into a standard legible form, according to its USAGE.|

12. The contents of DEBUG-LINE is the internal line number given at compilation time to any particular source statement, as shown in the compilation listing.

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

    All qualifiers of the name are separated in DEBUG-NAME by the word 'IN' or 'OF'. Subscripts/indices, if any, are not entered into DEBUG-name-

14. If the reference to a data item that causes the debugging section to be executed is subscripted or indexed, the occurrence number of each level is entered in DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3 respectively as necessary.

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

16. If the first execution of the first non-declarative 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'.

17. 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.

18. 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.

19. If reference to procedure-name-1 in the INPUT or OUTPUT phrase of a SORT or MERGE statement causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the SORT or MERGE statement that references procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains:

       (i) If the reference to procedure-name-1 is in the INPUT phrase of a SORT statement, 'SORT INPUT'.

       (ii) If the reference to procedure-name-1 is in the OUTPUT phrase of a SORT statement, 'SORT OUTPUT'.

       (iii) If the reference of procedure-name-1 is in the OUTPUT phrase of a MERGE statement, 'MERGE OUTPUT'.

20. If the transfer of control from the control mechanism associated with a PERFORM statement caused 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'.

21. 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'.

22. 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'.

23. If references to file-name-1 or cd-name-1 causes the debugging section to be executed, then:

    a. DEBUG-LINE identifies the source statement that references file-name-1, or cd-name-1.

    b. DEBUG-NAME contains the name of file-name-1 or cd-name-1.

    c. For READ, DEBUG-CONTENTS contains the entire record read.

    d. For all other references to file-name-1, DEBUG-CONTENTS contains spaces.

    e. For any reference to cd-name-1, DEBUG-CONTENTS contains the contents of the area associated with the cd-name.

24. If a reference to identifier-1 causes the debugging section to be executed, then:

    a. DEBUG-LINE identifies the source statement that references identifier-1,

    b. DEBUG-NAME contains the name of identifier-1, and

    c. DEBUG-CONTENTS contains the content of the data item referenced by identifier-1 at the time that control passes to the debugging section (see "General Rules" 5 and 6).

16.6 DEBUGGING LINES

A debugging line is any line with a 'D'|, or 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.

After all COPY and REPLACE statements have been processed, 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 |, unless the DEBUGMD parameter is specified in the $COBOL JCL statement, or if the NDEBUGMD parameter is specified.|

Successive debugging lines are allowed.

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

17. Reference Format

17.1 GENERAL DESCRIPTION

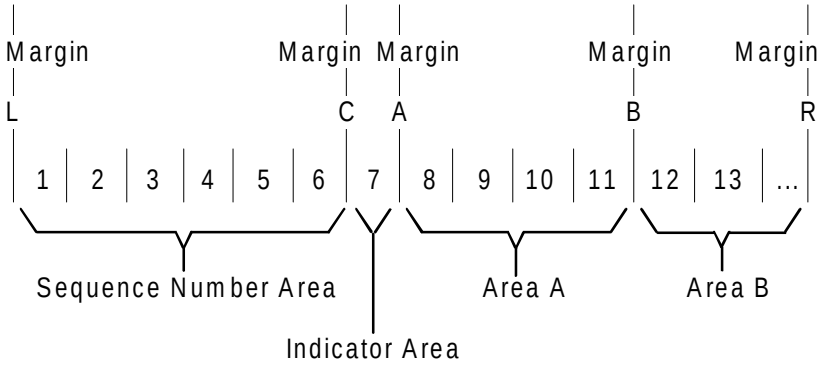
The reference format which provides a standard method for describing COBOL source programs and COBOL library texts, is described in terms of character positions in a line on an input-output medium. The COBOL compiler accepts source programs written in reference format and produces an output listing of the source programs 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 COBOL source program must be ordered as follows: |the Control Division, then| 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.

17.2 REFERENCE FORMAT REPRESENTATION

The reference format for a line is represented as follows:



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 72nd 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 position 8, 9, 10 and 11, and is between Margin A and Margin B.

Area B begins immediately to the right of Margin B and terminates immediately to the left of Margin R. It occupies a finite number of character positions.

|When 'Horizontal Tabulation' or 'BackSpace' characters are used in a position of a line before the 12th position, the following steps are to be performed in order, before the above rules apply:

- If there is a 'Horizontal Tabulation' character in one of the positions 1 to 6 of the line, the characters following the 'Horizontal Tabulation' character are moved to positions 8 and following, and positions from that containing the 'Horizontal Tabulation' character to position 7 inclusive are filled in with spaces;

- If there is a 'Horizontal Tabulation' character in position 7 of the line, the characters following the 'Horizontal Tabulation' character are moved to positions 12 and following, and positions 7 to 11 inclusive are filled in with spaces;

- If there is a 'BackSpace' character in position 7 or 8 of the line, the characters following the 'BackSpace' character are moved to positions 7 and following;

- If there is a 'Horizontal Tabulation' character in one of the positions 8 to 11 of the line, the characters following the 'Horizontal Tabulation' character are moved to positions 12 and following, and positions from that containing the 'Horizontal Tabulation' character to position 11 inclusive are filled in with spaces.|

17.2.1 Sequence Numbers

The sequence number area may be used to label a source program line. The content of the sequence number area is defined by the user and may consist of any character in the computer's character set. There is no requirement that the content of the sequence number area appear in a particular sequence or be unique. |However, the sequence of the sequence numbers may be checked according to a JCL option.|

|Note that when the format is COBOL or COBOLX, the sequence number area does not exist. Character positions given above are with regards to the reference format (including a possibly conceptual sequence number).|

17.2.2 Continuation of Lines

Any sentence, entry, phrase or clause may be continued by starting subsequent line(s) in Area B. These subsequent lines are called continuation line(s). The line being continued is called the continued line. Any word, literal or PICTURE character-string 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 non-blank character in Area B of the current line is the successor of the last non-blank character of the preceding line, excluding intervening comment lines or blank lines, without any intervening space. However, if the continued line contains a non-numeric literal without a closing quotation mark, the first non-blank character in Area B of 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 the continuation line must be blank.

If there is no hyphen in the Indicator Area of a line, it is assumed that the first non-blank character in the line is preceded by a space.

Both characters composing the separator "==" must be on the same line.

17.2.3 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. See "Continuation of Lines" above.

17.2.4 Comment Lines

A comment line is any line with an asterisk (*) or slant (/) in the indicator area of the line. A comment line may appear as any line in a source program after the Identification Division header. Any combination of the characters from the computer's character set may be included in Area A and Area B of that line. The asterisk or slant and the characters in Area A and Area B will be produced on the listing but serve as documentation only and will not be checked syntactically. The slant in the indicator area cause page ejection prior to printing the comment line in the listing of the source program; an asterisk in the indicator area causes printing of the line at the next available position in the listing.

Successive comment lines are allowed. There may be comment lines between a continued line and the related continuation line.

17.2.5 Pseudo-Texts

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

17.3 DIVISION, SECTION AND PARAGRAPH FORMATS

17.3.1 Division Header

The division header must start in Area A.

17.3.2 Section Header

The section header must start in Area A.

A section consists of zero, one or more paragraphs in the Environment Division or Procedure Division, or zero, one or more entries in the |Control Division or| Data Division.

17.3.3 Paragraph Header, Paragraph-name and Paragraph

A paragraph consists of a paragraph-name followed by the separator period and by zero, one or more sentences, or a paragraph header followed by one or more entries.

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 non-blank 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 non-blank line that is not a comment line.

When the sentences or entries of a paragraph require more than one line, they may be continued as described above. (See "Continuation of Lines" above)

17.4 DATA DIVISION ENTRIES

Each Data Division entry begins with a level indicator or a level-number followed by a space, followed by the name of the subject of the entry, followed by a sequence of independent clauses describing the item. The last clause is always terminated by a separator period.

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 any of the following: FD, SD, RD, CD.

In those entries that begin with a level indicator, the level indicator begins in Area A, followed by at least one space, and then followed with the name of the subject of entry and appropriate descriptive information.

Those entries that begin with level-numbers are called Data Description entries.

A level-number has a value taken from the set of values 01 through 49, 66, 77, 88. Level-numbers in the range 01 through 09 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 a level-number 01 or 77, the level-number begins in Area A followed by at least one space, and then followed with its associated record-name or item-name, if specified, and appropriate descriptive information.

Successive Data Description entries may be indented according to level-number. Any indentation is with respect to margin A. Each new Data Description entry may begin any number of positions to the right of margin A, except Data Description entries that begin with level 01 or 77 must begin in Area A. The extend of indentation is determined only by the width of the physical medium. 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.

17.5 DECLARATIVES

The key word DECLARATIVES and the combined key words END DECLARATIVES that precede and follow, respectively, the Declaratives portion of the Procedure Division must each appear on a line by itself. Each must begin in Area A and be followed by the separator period.

17.6 END PROGRAM HEADER

The End Program Header must start in Area A.

18. Intrinsic Functions

18.1 INTRODUCTION

18.1.1 Purpose of Intrinsic Function Module

The Intrinsic Function module provides the capability to reference a data item whose value is derived automatically at the time of reference during the execution of the object program.

18.1.2 Language Concepts

18.1.2.1 Function-Name

In the Intrinsic Function module, a function is a temporary data item whose value is determined by invoking a mechanism at the time the function is referenced during the execution of a statement. A function-name names a mechanism to determine the value of a function. A function-name is a COBOL word that is one of a specified list of COBOL words which may be used in COBOL source programs. See "Definitions of Functions" in this chapter.

18.1.2.2 Value Returned by a Function

The value returned by a function is considered to be a data value. A mechanism is provided at object time to assign a data value to a function when it is referenced. In order to determine the function's value, the evaluation mechanism may require access to data values provided by the referencing program. These data values are provided by specifying parameters, known as arguments, when referencing the function. Specific functions may place constraints on these arguments, such as range, etc. If, at the time a function is referenced, the arguments specified for that reference do not have values that comply with the specified constraints, the returned value for the function is undefined |unless specified in the "Returned Values" paragraph of the function description or an abnormal return code is returned by the Mathematical Package.|

18.1.2.3 Function-Identifier

A function-identifier is used by the programmer to reference a function within the Procedure Division of a COBOL source program. See "Function-Identifier" in Chapter 3.

18.2 GENERAL DESCRIPTION

18.2.1 Function Definition and Returned Value

Definition

The definition of a function identifies:

1. For alphanumeric functions, the size of the returned value.

2. For numeric and integer functions, the sign of the returned value and whether the function is integer.

3. For some other cases, the value returned.

Date Conversion Function

The Gregorian calendar is used in the date conversion functions. The starting date of Monday, January 1 1601, was chosen to establish a simple relationship between the Standard Date and DAY-OF-WEEK, that is, integer date 1 was a Monday, DAY-OF-WEEK 1.

18.2.2 Arguments

Definition

Arguments specify values used in the evaluation of a function. Arguments are specified in the function-identifier. These arguments can be specified as identifiers, as arithmetic expressions, or as literals. The definition of a function specifies the number of arguments required, which can be zero, one, or more. For some functions, the number of arguments which can be specified may be variable. The order in which arguments are specified in a function-identifier determines the interpretation given to each value in arriving at the function value.

Argument Types

Arguments may be required to have a certain class or a subset of a certain class. The types of argument are:

Numeric	An arithmetic expression must be specified. The value of the arithmetic expression, including operational sign, is used in determining the value of the function.
Alphabetic	An elementary data item of the class alphabetic or a non-numeric literal containing only alphabetic characters must be

specified. The size associated with the argument can be used in determining the value of the function.

Alphanumeric — A data item of the class alphabetic or alphanumeric or a non-numeric literal must be specified. The size associated with the argument can be used in determining the value of the function.

Integer — An arithmetic expression which will always result in an integer value must be specified. The value of the arithmetic expression, including operational sign, is used in determining the value of the function.

Permissible Values of Arguments

The rules for a function may place constraints on the permissible values for arguments in order to permit meaningful determination of the function's value. If, at the time a function is referenced, the arguments specified for that reference do not have values within the permissible range, the returned value for the function is undefined |unless specified in the "Returned Values" paragraph of the function description or an abnormal return code is returned by the Mathematical Package.|

Subscripting Using the Word ALL

When the definition of a function permits an argument to be repeated a variable number of times, a table may be referenced by specifying the data-name and any qualifiers that identify the table, followed immediately by subscripting where one or more of the subscripts is the word ALL.

When ALL is specified as a subscript, the effect is as if each table element associated with that subscript position were specified. The order of the implicit specification of each occurrence is from left to right, with the first (or leftmost) specification being the identifier with each subscript specified by the word ALL replaced by one, the next specification being the same identifier with the rightmost subscript specified by the word ALL incremented by one. This process continues with the rightmost ALL subscript being incremented by one for each implicit specification until the rightmost ALL subscript has been incremented through its range of values. If there are any additional ALL subscripts, the ALL subscript immediately to the left of the rightmost ALL subscript is incremented by one, the rightmost ALL subscript is reset to one and the process of varying the rightmost ALL subscript is repeated. The ALL subscript to the left of the rightmost ALL subscript is incremented by one through its range of values. For each additional ALL subscript, this process is repeated in turn until the leftmost ALL subscript has been incremented by one through its range of values. If the ALL subscript is associated with an OCCURS DEPENDING ON clause, the range of values is determined by the object of the OCCURS DEPENDING ON clause. The evaluation of an ALL subscript must result in at least one argument, otherwise the returned value is undefined.

18.3 TYPES OF FUNCTIONS

Data item functions are elementary data items and return alphanumeric, numeric, or integer values. Data item functions are treated as elementary data items and cannot be receiving operands.

The types of data item functions are:

Alphanumeric Functions

These are of the class and category alphanumeric. The number of character positions in this data item is specified in the function definition. Alphanumeric functions have an implicit usage of DISPLAY.

Numeric Functions

These are of the class and category numeric. A numeric function is always considered to have an operational sign. Those characteristics of the returned value not otherwise specified for a given function are defined by the "arithmetic expression intermediate results rules" (see Chapter 10).

A numeric function may be used only in an arithmetic expression.

A numeric function may not be referenced where an integer operand is required, even though a particular reference may yield an integer value.

Integer Functions

These are of the class and category numeric. An integer function is always considered to have an operational sign. Those characteristics of the returned value not otherwise specified for a given function are defined by the "arithmetic expression intermediate results rules" (see Chapter 10).

An integer function may be used only in an arithmetic expression.

An integer function can be referenced where an integer operand is required and where a signed operand is allowed.

18.4 DEFINITION OF FUNCTIONS

The "Table of Functions" below summarized the functions that are available.
The Arguments column defines the type and number of arguments as follows:
A means alphabetic.
I means integer.
N means numeric.
X means alphanumeric.

The Type column defines the type of function as follows:
I means integer.
N means numeric.
X means alphanumeric.

Table 18-1. Table of Functions (1/3)

Function-Name	Arguments	Type	Value Returned
ACOS	N1	N	Arccosine of N1
ANNUITY	N1, I2	N	Ratio of annuity paid for I2 periods at interest of N1 to an initial investment of one
ASIN	N1	N	Arcsine of N1
ATAN	N1	N	Arctangent of N1
CHAR	I1	X	Character in position I1 of program collating sequence
COS	N1	N	Cosine of N1
CURRENT-DATE	None	X	Current date and time and the difference from Greenwich Mean Time
DATE-OF-INTEGER	I1	I	Standard date equivalent (YYYYMMDD) of integer date
DAY-OF-INTEGER	I1	I	Julian date equivalent (YYYYDDD)of integer date
FACTORIAL	I1	I	Factorial of I1
INTEGER	N1	I	The greatest integer not greater than N1
INTEGER-OF-DATE	I1	I	Integer date equivalent of standard date (YYYYMMDD)

Table 18-1. Table of Functions (2/3)

Function-Name	Arguments	Type	Value Returned
INTEGER-OF-DAY	I1	I	Integer date equivalent of Julian date (YYYYDDD)
INTEGER-PART	N1	I	Integer part of N1
LENGTH	A1, N1, or X1	I	Length of argument
LOG	N1	N	Natural logarithm of N1
LOG10	N1	N	Logarithm to base 10 of N1
LOWER-CASE	A1 or X1	X	All letters in the argument are set to lower-case
MAX	A1 ... or I1 ... or N1 ... or X1 ...	Depends on arguments (a function that has only alphabetic arguments is type alphanumeric)	Value of maximum argument
MEAN	N1	N	Arithmetic mean of arguments
MEDIAN	N1	N	Median of arguments
MIDRANGE	N1	N	Mean of minimum and maximum arguments
MIN	A1 ... or I1 ... or N1 ... or X1 ...	Depends on arguments (a function that has only alphabetic arguments is type alphanumeric)	Value of minimum argument
MOD	I1, I2	I	I1 modulo I2
NUMVAL	X1	N	Numeric value of numeric simple string
NUMVAL-C	X1, X2	N	Numeric value of numeric string with optional commas and currency sign

Table 18-1. Table of Functions (3/3)

Function-Name	Arguments	Type	Value Returned
ORD	A1 or X1	I	Ordinal position of the argument in collating sequence
ORD-MAX	A1 ... or N1 ... or X1 ...	I	Ordinal position of maximum argument
ORD-MIN	A1 ... or N1 ... or X1 ...	I	Ordinal position of minimum argument
PRESENT-VALUE	N1 N2 ...	N	Present value of a series of future period-end amounts, N2, at a discount rate of N1
RANDOM	I1	N	Random number
RANGE	I1 ... or N1 ...	Depends on arguments	Value of maximum argument minus value of minimum argument
REM	N1, N2	N	Remainder of N1/N2
REVERSE	A1 or X1	X	Reverse order of the characters in the argument
SIN	N1	N	Sine of N1
SQRT	N1	N	Square root of N1
STANDARD-DEVIATION	N1	N	Standard deviation of arguments
SUM	I1 ... or N1	Depends on arguments	Sum of arguments
TAN	N1	N	Tangent of N1
UPPER-CASE	A1 or X1	X	All letters in the argument are set to upper-case
VARIANCE	N1	N	Variance of argument
WHEN-COMPILED	None	X	Date and time the program was compiled

18.5 ACOS FUNCTION

Description

The ACOS function returns a numeric value in radians that approximates the arccosine of argument-1.

Type

The type of this function is numeric.

General Format

```
FUNCTION ACOS (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

2. The value of argument-1 must be greater than or equal to -1 and less than or equal to +1.

Returned Values

1. The returned value is the approximation of the arccosine of argument-1 and is greater than or equal to zero and less than or equal to pi.

18.6 ANNUITY FUNCTION

Description

The ANNUITY function (annuity immediate) returns a numeric value that approximates the ratio of an annuity paid at the end of each period for the number of periods specified by argument-2 to an initial investment of one. Interest is earned at the rate specified by argument-1 and is applied at the end of the period, before the payment.

Type

The type of this function is numeric.

General Format

```
FUNCTION ANNUITY (argument-1 argument-2)
```

Arguments

1. Argument-1 must be class numeric.
2. The value of argument-1 must be greater than or equal to zero.
3. Argument-2 must be a positive integer.

Returned Values

1. When the value of argument-1 is zero, the value of the function is the approximation of:

```
1 / argument-2
```

2. When the value of argument-1 is not zero, the value of the function is the approximation of:

```
argument-1 / (1 - (1 + argument-1) ** (- argument-2))
```

18.7 ASIN FUNCTION

Description

The ASIN function returns a numeric value in radians that approximates the arcsine of argument-1.

Type

The type of this function is numeric.

General Format

```
FUNCTION ASIN (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

2. The value of argument-1 must be greater than or equal to -1 and less than or equal to +1.

Returned Values

1. The returned value is the approximation of the arcsine of argument-1 and is greater than or equal to -pi/2 and less than or equal to +pi/2.

18.8 ATAN FUNCTION

Description

The ATAN function returns a numeric value in radians that approximates the arctangent of argument-1.

Type

The type of this function is numeric.

General Format

```
FUNCTION ATAN (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

Returned Values

1. The returned value is the approximation of the arctangent of argument-1 and is greater than -pi/2 and less than +pi/2.

18.9 CHAR FUNCTION

Description

The CHAR function returns a one-character alphanumeric value that is a character in the program collating sequence having the ordinal position equal to the value of argument-1.

Type

The type of this function is alphanumeric.

General Format

```
FUNCTION CHAR (argument-1)
```

Arguments

1. Argument-1 must be an integer.

2. The value of argument-1 must be greater than zero and less than or equal to the number of positions in the collating sequence.

Returned Values

1. If more than one character has the same position in the program collating sequence, the character returned as the function value is that of the first literal specified for that character position in the ALPHABET clause.

2. If the current program collating sequence was not specified by an ALPHABET clause, the collating sequence is the default collating sequence as specified in the OBJECT-COMPUTER general rules. See OBJECT-COMPUTER, Chapter 7.

18.10 COS FUNCTION

Description

The COS function returns a numeric value that approximates the cosine of an angle or arc, expressed in radians, that is specified by argument-1.

Type

The type of this function is numeric.

General Format

```
FUNCTION COS (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

Returned Values

1. The returned value is the approximation of the cosine of argument-1 and is greater than or equal to -1 and less than or equal to +1.

18.11 CURRENT-DATE FUNCTION

Description

The CURRENT-DATE function returns a 21-character alphanumeric value that represents the calendar date, time of day, and local time differential factor provided by the system on which the function is evaluated.

Type

The type of this function is alphanumeric.

General Format

```
FUNCTION CURRENT-DATE
```

Arguments

None.

Returned Values

1. The character positions returned, numbered from left to right, are:

Character Positions	Contents
1-4	Four numeric digits of the year in the Gregorian calendar.
5-6	Two numeric digits of the month of the year, in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range 00 through 23.
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric digits of the seconds past the minute, in the range 00 through 59.
15-16	Two numeric digits of the hundredths of a second past the second, in the range 00 through 99. The value 00 is returned if the system on which the function is evaluated does not have the facility to provide the fractional part of a second.

Character Positions	Contents
17	Either the character '-', the character '+', or the character '0'. The character '-' is returned if the local time indicated in the previous character positions is behind Greenwich Mean Time. The character '+' is returned if the local time indicated is the same as or ahead of Greenwich Mean Time. The character '0' is returned if the system on which this function is evaluated does not have the facility to provide the local time differential factor.
18-19	If character position 17 is '-', two numeric digits are returned in the range 00 through 12 indicating the number of hours that the reported time is behind Greenwich Mean Time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the reported time is ahead of Greenwich Mean Time. If character position 17 is '0', the value 00 is returned.
20-21	Two numeric digits are returned in the range 00 through 59 indicating the number of additional minutes that the reported time is ahead of or behind Greenwich Mean Time, depending on whether character position 17 is '+' or '-', respectively. If character position 17 is '0', the value 00 is returned.

2. If the system does not have the facility to provide fractional parts of a second, the value 00 is returned in character positions 15 and 16.

3. If the system does not have the facility to provide the local time differential factor, the value 00000 is returned in character positions 17 through 21.

18.12 DATE-OF-INTEGER FUNCTION

Description

The DATE-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to standard date form (YYYYMMDD).

Type

The type of this function is integer.

General Format

```
FUNCTION DATE-OF-INTEGER (argument-1)
```

Arguments

1. Argument-1 is a positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar.

Returned Values

1. The returned value represents the ISO Standard date of the integer specified in argument-1.

2. The returned value is in the form (YYYYMMDD), where YYYY represents a year in the Gregorian calendar, MM represents the month of that year, and DD represents the day of that month.

|3. The returned value is 0 (zero) if argument-1 is less than 1.|

18.13 DAY-OF-INTEGER FUNCTION

Description

The DAY-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to Julian date form (YYYYDDD).

Type

The type of this function is integer.

General Format

```
FUNCTION DAY-OF-INTEGER (argument-1)
```

Arguments

1. Argument-1 is a positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar.

Returned Values

1. The returned value represents the Julian equivalent of the integer specified in argument-1.

2. The returned value is an integer of the form (YYYYDDD), where YYYY represents a year in the Gregorian calendar, and DDD represents the day of that year.

|3. The returned value is 0 (zero) if argument-1 is less than 1.|

18.14 FACTORIAL FUNCTION

Description

The FACTORIAL function returns an integer that is the factorial of argument-1.

Type

The type of this function is integer.

General Format

```
FUNCTION FACTORIAL (argument-1)
```

Arguments

1. Argument-1 must be an integer greater than or equal to zero.

Returned Values

1. If the value of argument-1 is zero, the value 1 is returned.
2. If the value of argument-1 is positive, its factorial is returned.

18.15 INTEGER FUNCTION

Description

The INTEGER function returns the greatest integer value that is less than or equal to argument-1.

Type

The type of this function is integer.

General Format

```
FUNCTION INTEGER (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

Returned Values

1. The returned value is the greatest integer less than or equal to the value of argument-1. For example, if the value of argument-1 is -1.5, -2 is returned; if the value of argument-1 is +1.5, +1 is returned.

18.16 INTEGER-OF-DATE FUNCTION

Description

The INTEGER-OF-DATE function converts a date in the Gregorian calendar from standard date form (YYYYMMDD) to integer date form.

Type

The type of this function is integer.

General Format

```
FUNCTION INTEGER-OF-DATE (argument-1)
```

Arguments

1. Argument-1 must be an integer of the form YYYYMMDD, whose value is obtained from the calculation:

```
(YYYY * 10,000) + (MM * 100) + DD
```

   a. YYYY represents the year in the Gregorian calendar. It must be an integer greater than 1600.

   b. MM represents a month and must be a positive integer less than 13.

   c. DD represents a day and must be a positive integer less than 32 provided that it is valid for the specified month and year combination.

Returned Values

1. The returned value is an integer that is the number of days the date represented by argument-1 succeeds December 31, 1600, in the Gregorian calendar.

|2. The returned value is 0 (zero) if argument-1 contains illegal values for year, month or day.|

18.17 INTEGER-OF-DAY FUNCTION

Description

The INTEGER-OF-DAY function converts a date in the Gregorian calendar from Julian date form (YYYYDDD) to integer date form.

Type

The type of this function is integer.

General Format

```
FUNCTION INTEGER-OF-DAY (argument-1)
```

Arguments

1. Argument-1 must be an integer of the form YYYYDDD, whose value is obtained from the calculation:

```
(YYYY * 1000) + DDD
```

   a. YYYY represents the year in the Gregorian calendar. It must be an integer greater than 1600.

   b. DDD represents the day of the year. It must be a positive integer less than 367 provided that it is valid for the year specified.

Returned Values

1. The returned value is an integer that is the number of days the date represented by argument-1 succeeds December 31, 1600, in the Gregorian calendar.

|2. The returned value is 0 (zero) if argument-1 contains illegal values for year, month or day.|

18.18 INTEGER-PART FUNCTION

Description

The INTEGER-PART function returns an integer that is the integer portion of argument-1.

Type

The type of this function is integer.

General Format

```
FUNCTION INTEGER-PART (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

Returned Values

1. If the value of argument-1 is zero, the returned value is zero.

2. If the value of argument-1 is positive, the returned value is the greatest integer less than or equal to the value of argument-1.

   For example, if the value of argument is +1.5, then +1 is returned.

3. If the value of argument-1 is negative, the returned value is the least integer greater than or equal to the value of argument-1.

   For example, if the value of argument is -1.5, then -1 is returned.

18.19 LENGTH FUNCTION

Description

The LENGTH function returns an integer equal to the length of the argument in character positions.

Type

The type of this function is integer.

General Format

```
FUNCTION LENGTH (argument-1)
```

Arguments

1. Argument-1 may be a non-numeric literal or a data item of any class or category.

2. If argument-1 or any data item subordinate to argument-1 is described with the DEPENDING phrase of the OCCURS clause, the contents of the data item referenced by the data-name specified in the DEPENDING phrase are used at the time the LENGTH function is evaluated.

3. If argument-1 is described with the DEPENDING phrase of the PICTURE clause, the contents of the data item referenced by the data-name specified in the DEPENDING phrase are used at the time the LENGTH function is evaluated.

Returned Values

1. If argument-1 is a non-numeric literal or an elementary data item, or argument-1 is a group data item that does not contain a variable occurrence data item, the value returned is an integer equal to the length of argument-1 in character positions.

2. If argument-1 is a group data item containing a variable occurrence data item, the value returned is an integer determined by evaluation of the data item specified in the DEPENDING phrase of the OCCURS clause for that variable occurrence data item. This evaluation is accomplished according to the rules in the OCCURS clause dealing with the data item as a sending data item. See the "OCCURS Clause" and the "USAGE Clause" in Chapter 9.

3. If argument-1 is an elementary data item described with the DEPENDING phrase of the PICTURE clause, the value returned is that of the data item referenced by the data-name specified in the DEPENDING phrase.

4. The returned value includes implicit FILLER characters, if any.

18.20 LOG FUNCTION

Description

The LOG function returns a numeric value that approximates the logarithm to the base e (natural log) of argument-1.

Type

The type of this function is numeric.

General Format

```
FUNCTION LOG (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

2. The value of argument-1 must be greater than zero.

Returned Values

1. The returned value is the approximation of the logarithm to the base of argument-1.

18.21 LOG10 FUNCTION

Description

The LOG10 function returns a numeric value that approximates the logarithm to the base 10 of argument-1.

Type

The type of this function is numeric.

General Format

```
FUNCTION LOG10 (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

2. The value of argument-1 must be greater than zero.

Returned Values

1. The returned value is the approximation of the logarithm to the base 10 of argument-1.

18.22 LOWER-CASE FUNCTION

Description

The LOWER-CASE function returns a character string that is the same length as argument-1 with each upper-case letter replaced by the corresponding lower-case letter.

Type

The type of this function is alphanumeric.

General Format

```
FUNCTION LOWER-CASE (argument-1)
```

Arguments

1. Argument-1 must be class alphabetic or alphanumeric and must be at least one character in length.

Returned Values

1. The same character string as argument-1 is returned, except that each upper-case letter is replaced by the corresponding lower-case letter.

2. The character string returned has the same length as argument-1.

3. If the computer character set does not include lower-case letters, no changes take place in the character string.

18.23 MAX FUNCTION

Description

The MAX function returns the content of the argument-1 that contains the maximum value.

Type

The type of this function depends on the argument types as follows:

Argument Type	Function Type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

General Format

```
FUNCTION MAX ({argument-1} ... )
```

Arguments

1. If more than one argument-1 is specified, all arguments must be of the same class.

Returned Values

1. The returned value is the content of the argument-1 having the greatest value. The comparisons used to determine the greatest value are made according to the rules for simple conditions (see Chapter 10).

2. If more than one argument-1 has the same greatest value, the content of the argument-1 returned is the leftmost argument-1 having that value.

3. If the type of the function is alphanumeric, the size of the returned value is the same as the size of the selected argument-1.

18.24 MEAN FUNCTION

Description

The MEAN function returns a numeric value that is the arithmetic mean (average) of its arguments.

Type

The type of this function is numeric.

General Format

```
FUNCTION MEAN ({argument-1} ... )
```

Arguments

1. Argument-1 must be class numeric.

Returned Values

1. The returned value is the arithmetic mean of the argument-1 series.

2. The returned value is defined as the sum of the argument-1 series divided by the number of occurrences referenced by argument-1.

18.25 MEDIAN FUNCTION

Description

The MEDIAN function returns the content of the argument whose value is the middle value in the list formed by arranging the arguments in sorted order.

Type

The type of this function is numeric.

General Format

```
FUNCTION MEDIAN ({argument-1} ... )
```

Arguments

1. Argument-1 must be class numeric.

Returned Values

1. The returned value is the content of the argument-1 having the middle value in the list formed by arranging all the argument-1 values in sorted order.

2. If the number of occurrences referenced by argument-1 is odd, the returned value is such that at least half of the occurrences referenced by argument-1 are greater than or equal to the returned value and at least half are less than or equal. If the number of occurrences referenced by argument-1 is even, the returned value is the arithmetic mean of the values referenced by the two middle occurrences.

3. The comparisons used to arrange the argument-1 values in sorted order are made according to the rules for simple conditions (see Chapter 10).

18.26 MIDRANGE FUNCTION

Description

The MIDRANGE (middle range) function returns a numeric value that is the arithmetic mean (average) of the values of the minimum argument and the maximum argument.

Type

The type of this function is numeric.

General Format

```
FUNCTION MIDRANGE ({argument-1} ... )
```

Arguments

1. Argument-1 must be class numeric.

Returned Values

1. The returned value is the arithmetic mean of the greatest argument-1 value and the least argument-1 value. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions (see Chapter 10).

18.27 MIN FUNCTION

Description

The MIN function returns the content of the argument-1 that contains the minimum value.

Type

The type of this function depends on the argument types as follows:

Argument Type	Function Type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

General Format

```
FUNCTION MIN ({argument-1} ... )
```

Arguments

1. If more than one argument-1 is specified, all arguments must be of the same class.

Returned Values

1. The returned value is the content of the argument-1 having the least value. The comparisons used to determine the least value are made according to the rules for simple conditions (see Chapter 10).

2. If more than one argument-1 has the same least value, the content of the argument-1 returned is the leftmost argument-1 having that value.

3. If the type of the function is alphanumeric, the size of the returned value is the same as the size of the selected argument-1.

18.28 MOD FUNCTION

Description

The MOD function returns an integer value that is argument-1 modulo argument-2.

Type

The type of this function is integer.

General Format

```
FUNCTION MOD (argument-1 argument-2)
```

Arguments

1. Argument-1 and argument-2 must be integers.
2. The value of argument-2 must not be zero.

Returned Values

1. The returned value is argument-1 modulo argument-2.

   The returned value is defined as:

   argument-1 - (argument-2 * FUNCTION INTEGER (argument-1 / argument-2))

2. The following illustrates the expected results for some values of argument-1 and argument-2.

Argument-1	Argument-2	Return
11	5	1
-11	5	4
11	-5	-4
-11	-5	-1

18.29 NUMVAL FUNCTION

Description

The NUMVAL function returns the numeric value represented by the character string specified by argument-1. Leading and trailing spaces are ignored.

Type

The type of this function is numeric.

General Format

```
FUNCTION NUMVAL (argument-1)
```

Arguments

1. Argument-1 must be a non-numeric literal or alphanumeric data item whose content has one of the following two formats:

```
           [+]         {digit [ . [digit]]}
- [space] [ ] [space] {                   } [space]
           [-]         {. digit            }
```

or

```
          {digit [ . [digit]]}         [+ ]
          {                   }        [- ]
[space]   {                   } [space] [  ] [space]
          {. digit            }         [CR]
          {                   }         [DB]
```

where space is a string of zero or more spaces and digit is a string of one to 18 digits.

2. The total number of digits in argument-1 must not exceed 18.

3. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, a comma must be used in argument-1 rather than a decimal point.

Returned Values

1. The returned value is the numeric value represented by argument-1.

2. The number of digits returned is 18.

18.30 NUMVAL-C FUNCTION

Description

The NUMVAL-C function returns the numeric value represented by the character string specified by argument-1. Any optional currency sign specified by argument-2 and any optional commas preceding the decimal point are ignored.

Type

The type of this function is numeric.

General Format

```
FUNCTION NUMVAL-C (argument-1 [argument-2])
```

Arguments

1. Argument-1 must be a non-numeric literal or alphanumeric data item whose content has one of the following two formats:

```
                                {digit [, digit]      }
        [+]                     {      ... [. [digit]]}
[space] [ ] [space] [cs] [space] {                     } [space]
        [-]                     {. digit               }
```

or

```
                      {digit [, digit]      }
                      {      ... [. [digit]]}         [+ ]
                      {                     }         [- ]
[space] [cs]  [space] {                     } [space] [  ] [space]
                      {. digit              }         [CR]
                      {                     }         [DB]
```

   where space is a string of zero or more spaces, cs is the string of one or more characters specified by argument-2, and digit is a string of one or more digits.

2. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, the functions of the comma and decimal point in argument-1 are reversed.

3. The total number of digits in argument-1 must not exceed 18.

4. Argument-2, if specified, must be a non-numeric literal or alphanumeric data item.

5. If argument-2 is not specified, the character used for cs is the currency symbol specified for the program.

Returned Values

1. The returned value is the numeric value represented by argument-1.

2. The number of digits returned is 18.

18.31 ORD FUNCTION

Description

The ORD function returns an integer value that is the ordinal position of argument-1 in the collating sequence for the program. The lowest ordinal position is 1.

Type

The type of this function is integer.

General Format

```
FUNCTION ORD (argument-1)
```

Arguments

1. Argument-1 must be one character in length and must be class alphabetic or alphanumeric.

Returned Values

1. The returned value is the ordinal position of argument-1 in the collating sequence for the program.

18.32 ORD-MAX FUNCTION

Description

The ORD-MAX function returns a value that is the ordinal number of the argument-1 that contains the maximum value.

Type

The type of this function is integer.

General Format

```
FUNCTION ORD-MAX ({argument-1} ... )
```

Arguments

1. If more than one argument-1 is specified, all arguments must be of the same class.

Returned Values

1. The returned value is the ordinal number that corresponds to the position of the argument-1 having the greatest value in the argument-1 series.

2. The comparisons used to determine the greatest valued argument are made according to the rules for simple conditions (see Chapter 10).

3. If more than one argument-1 has the same greatest value, the number returned corresponds to the position of the leftmost argument-1 having that value.

18.33 ORD-MIN FUNCTION

Description

The ORD-MIN function returns a value that is the ordinal number of the argument that contains the minimum value.

Type

The type of this function is integer.

General Format

```
FUNCTION ORD-MIN ({argument-1} ... )
```

Arguments

1. If more than one argument-1 is specified, all arguments must be of the same class.

Returned Values

1. The returned value is the ordinal number that corresponds to the position of the argument-1 having the least value in the argument-1 series.

2. The comparisons used to determine the least valued argument-1 are made according to the rules for simple conditions (see Chapter 10).

3. If more than one argument-1 has the same least value, the number returned corresponds to the position of the leftmost argument-1 having that value.

18.34 PRESENT-VALUE FUNCTION

Description

The PRESENT-VALUE function returns a value that approximates the present value of a series of future period-end amounts specified by argument-2 at a discount rate specified by argument-1.

Type

The type of this function is numeric.

General Format

```
FUNCTION PRESENT-VALUE (argument-1 {argument-2} ... )
```

Arguments

1. Argument-1 and argument-2 must be of the class numeric.

2. The value of argument-1 must be greater than -1.

Returned Values

1. The returned value is an approximation of the summation of a series of calculations with each term in the following form:

   ```
   argument-2 / (1 + argument-1) ** n
   ```

   There is one term for each occurrence of argument-2. The exponent, n, is incremented from one by one for each term in the series.

18.35 RANDOM FUNCTION

Description

The RANDOM function returns a numeric value that is a pseudo-random number from a rectangular distribution.

Type

The type of this function is numeric.

General Format

```
FUNCTION RANDOM [(argument-1)]
```

Arguments

1. If argument-1 is specified, it must be zero or a positive integer. It is used as the seed value to generate a sequence of pseudo-random numbers.

2. If a subsequent reference specifies argument-1, a new sequence of pseudo-random numbers is started.

3. If the first reference to this function in the run unit does not specify argument-1, the seed value is 1.

4. In each case, subsequent references without specifying argument-1 return the next number in the current sequence.

Returned Values

1. The returned value is greater than or equal to zero and less than one.

2. For a given seed value, the sequence of pseudo-random numbers will always be the same.

3. The subset of the domain of argument-1 values that will yield distinct sequences of pseudo-random numbers is 0 through 2147483647.

18.36 RANGE FUNCTION

Description

The RANGE function returns a value that is equal to the value of the maximum argument minus the value of the minimum argument.

Type

The type of this function depends on the argument types as follows:

Argument Type	Function Type
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

General Format

```
FUNCTION RANGE ({argument-1} ... )
```

Arguments

1. Argument-1 must be class numeric.

Returned Values

1. The returned value is equal to the greatest value of argument-1 minus the least value of argument-1. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions (see Chapter 10).

18.37 REM FUNCTION

Description

The REM function returns a numeric value that is the remainder of argument-1 divided by argument-2.

Type

The type of this function is numeric.

General Format

```
FUNCTION REM (argument-1 argument-2)
```

Arguments

1. Argument-1 and argument-2 must be class numeric.

2. The value of argument-2 must not be zero.

Returned Values

1. The returned value is the remainder of argument-1 / argument-2.

   The returned value is defined as:

```
argument-1 - (argument-2 * FUNCTION INTEGER-PART
                                    (argument-1 / argument-2))
```

18.38 REVERSE FUNCTION

Description

The REVERSE function returns a character string of exactly the same length as argument-1 and whose characters are exactly the same as those of argument-1, except that they are in reverse order.

Type

The type of this function is alphanumeric.

General Format

```
FUNCTION REVERSE (argument-1)
```

Arguments

1. Argument-1 must be class alphabetic or alphanumeric and must be at least one character in length.

Returned Values

1. If argument-1 is a character string of length n, the returned value is a character string of length n such that for 1 <= j <= n, the character in position j of the returned value is the character from position n-j+1 of argument-1.

18.39 SIN FUNCTION

Description

The SIN function returns a numeric value that approximates the sine of an angle or arc, expressed in radians, that is specified by argument-1.

Type

The type of this function is numeric.

General Format

```
FUNCTION SIN (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

Returned Values

1. The returned value is the approximation of the sine of argument-1 and is greater than or equal to -1 and less than or equal to +1.

18.40 SQRT FUNCTION

Description

The SQRT function returns a numeric value that approximates the square root of argument-1.

Type

The type of this function is numeric.

General Format

```
FUNCTION SQRT (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

2. The value of argument-1 must be zero or positive.

Returned Values

1. The returned value is the absolute value of the approximation of the square root of argument-1.

18.41 STANDARD-DEVIATION FUNCTION

Description

The STANDARD-DEVIATION function returns a numeric value that approximates the standard deviation of its arguments.

Type

The type of this function is numeric.

General Format

```
FUNCTION STANDARD-DEVIATION ({argument-1} ... )
```

Arguments

1. Argument-1 must be class numeric.

Returned Values

1. The returned value is defined as the square of the standard deviation of the argument-1 series.

2. The returned value is calculated as follows:

   a. The difference between each argument-1 value and the arithmetic mean of the argument-1 series is calculate and squared.

   b. The values obtained are then added together. This quantity is divided by the number of values in the argument-1 series.

   c. The square root of the quotient obtained is then calculated. The returned value is the absolute value of this square root.

3. If the argument-1 series consists of only one value, or if the argument-1 series consists of all variable occurrence data items and the total number of occurrences for all of them is one, the returned value is zero.

18.42 SUM FUNCTION

Description

The SUM function returns a value that is sum of the arguments.

Type

The type of this function depends on the argument types as follows:

Argument Type	Function Type
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

General Format

```
FUNCTION SUM ({argument-1} ... )
```

Arguments

1. Argument-1 must be class numeric.

Returned Values

1. The returned value is the sum of the arguments.
2. If the argument-1 series are all integers, the value returned is an integer.
3. If the argument-1 series are not all integers, a numeric value is returned.

18.43 TAN FUNCTION

Description

The TAN function returns a numeric value that approximates the tangent of an angle or arc, expressed in radians, that is specified by argument-1.

Type

The type of this function is numeric.

General Format

```
FUNCTION TAN (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

Returned Values

1. The returned value is the approximation of the tangent of argument-1.

18.44 UPPER-CASE FUNCTION

Description

The UPPER-CASE function returns a character string that is the same length as argument-1 with each lower-case letter replaced by the corresponding upper-case letter.

Type

The type of this function is alphanumeric.

General Format

```
FUNCTION UPPER-CASE (argument-1)
```

Arguments

1. Argument-1 must be class alphabetic or alphanumeric and must be at least one character in length.

Returned Values

1. The same character string as argument-1 is returned, except that each lower-case letter is replaced by the corresponding upper-case letter.

2. The character string returned has the same length as argument-1.

18.45 VARIANCE FUNCTION

Description

The VARIANCE function returns a numeric value that approximates the variance of its arguments.

Type

The type of this function is numeric.

General Format

```
FUNCTION VARIANCE ({argument-1} ... )
```

Arguments

1. Argument-1 must be class numeric.

Returned Values

1. The returned value is the approximation of the variance of the argument-1 series.

2. The returned value is defined as the square of the standard deviation of the argument-1 series. See the paragraph "Returned Values" of the STANDARD-DEVIATION function in this chapter.

3. If the argument-1 series consists of only one value, or if the argument-1 series consists of all variable occurrence data items and the total number of occurrences for all of them is one, the returned value is zero.

18.46 WHEN-COMPILED FUNCTION

Description

The WHEN-COMPILED function returns the date and time the program was compiled as provided by the system on which the program was compiled.

Type

The type of this function is alphanumeric.

General Format

```
FUNCTION WHEN-COMPILED
```

Arguments

None.

Returned Values

1. The character positions returned, numbered from left to right, are:

Character	Contents
1-4	Four numeric digits of the year in the Gregorian calendar.
5-6	Two numeric digits of the month of the year, in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range 00 through 23.
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric digits of the seconds past the minute, in the range 00 through 59.
15-16	Two numeric digits of the hundredths of a second past the second, in the range 00 through 99. The value 00 is returned if the system on which the program was compiled did not have the facility to provide the fractional part of a second.

Character	Contents
17	Either the character '-', the character '+', or the character '0'. The character '-' is returned if the local time of compilation, reported in the previous character positions, is behind Greenwich Mean Time. The character '+' is returned if the reported time is the same as or ahead of Greenwich Mean Time. The character '0' is returned if the system on which the program was compiled did not have the facility to provide the local time differential factor.
18-19	If character position 17 is '-', two numeric digits are returned in the range 00 through 12 indicating the number of hours that the reported time is behind Greenwich Mean Time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the reported time is ahead of Greenwich Mean Time. If character position 17 is '0', the value 00 is returned.
20-21	Two numeric digits are returned in the range 00 through 59 indicating the number of additional minutes that the reported time is ahead of or behind Greenwich Mean Time, depending on whether character position 17 is '+' or '-', respectively. If character position 17 is '0', the value 00 is returned.

2. The returned value is the date and time of compilation of the source program that contains this function. If the program is a contained program, the returned value is the compilation date and time associated with the separately compiled program in which it is contained.

3. The returned value denotes the same time as the compilation date and time provided in the listing of the source program, although their representations and precisions may differ.

A. COBOL Reserved Words

The following pages contain an alphabetized master list of all key words (reserved or not) of the GCOS DPS 7 COBOL Compiler described in this manual. They include also words which are not reserved in the current implementation but which are used in the CODASYL COBOL Journal Of Development, and as such are candidates to become reserved words in a next revision of the Standard. Reserved words of the Data Manipulation Language (DML) are listed too. The notations in the columns to the right of the list indicate the reason for the inclusion of each word.

In col.1 the notation 'DPS7' indicates that the word is a DPS 7 reserved word. (See note below). Only these words are processed as reserved words by the GCOS DPS 7 COBOL Compiler.

In col.1 the notations '(1)', '(2)', '(3)', '(4)', '(5)', '(6)', '(7)', and '(8)' also indicate that the word is, or will be, a DPS7 key word, with the following qualification:

1) the word must not be used as a User-Defined Word if it is referenced in the PROGRAM COLLATING SEQUENCE clause, or in a CODE-SET clause, or in the COLLATING SEQUENCE phrase of a SORT or MERGE statement (except if it is defined as an alphabet-name).

2) the word is a DML reserved word, i.e. it is reserved only when the DML facility is used; otherwise, it may be used as a User-Defined Word though it is a DPS 7 COBOL key word.

3) the word must not be used as a User-Defined Word if it is referenced in the FROM phrase of an ACCEPT statement, the UPON phrase of a DISPLAY statement, or in a SET statement (except if it defined as a mnemonic-name which can be referenced in that context).

4) the word is a DML reserved word; i.e., it is reserved only when the DML facility is used; otherwise, it may be used as a User-Defined Word.

5) the word is a DML reserved word; i.e., it is reserved only when the DML facility is used; otherwise, it may be used as a User-Defined Word, but it will become a reserved word in another version of the compiler.

6) the word may be used as a User-Defined Word, but it will become a reserved word in another version of the compiler.

7) the word may be used as a User-Defined Word though it is a DPS 7 COBOL key word.

8) the word is a Function-Name; in a different context, it may appear in a program as a user-defined word or a system-name, and in this case cannot be used as a Function-Name.

Rules (1), (2), (3), (4), and (5) do not apply when the compilation level specified in the LEVEL parameter of the $COBOL JCL statement is DSA (see specific rule below).

In col.2 the notation 'DSA' indicates that the word is reserved by the DPS 7 COBOL Compiler when the compilation level is DSA In this case, any other word may be used as a User-Defined Word with no restriction. However, if such a word is a DPS 7 (completely or partially) reserved word, and it is used in a DSA COBOL program, that DSA COBOL program will not compile or execute correctly when the compilation level is other than DSA.

In col.3 the notation 'ANSI' indicates that the word is reserved by American National Standard COBOL specification X3.23-1985. It is therefore one of the DPS 7 reserved words.

In col.4 the notation 'JOD' indicates that the word is a reserved word in the CODASYL Journal Of Development updated January 1985. The notation 'SBS' in col.4 indicates that the word is a reserved word in the subschema description in the CODASYL Journal Of Development (as of January 1985).

When writing source language for a program to be compiled by the GCOS DPS 7 COBOL Compiler, the programmer should not use any of the listed words as a data-name, procedure-name, or for any other unreserved use. DPS 7 COBOL reserved words should be used only as reserved in this manual.

Notes: DPS 7 reserved words include:

1. Words specified as reserved in the format descriptions in this manual.

2. Words reserved for future implementation.

3. DPS Series COBOL compilers.

	1	2	3	4
.	DPS7	DSA	ANSI	JOD
<	DPS7	DSA	ANSI	JOD
<=	DPS7	DSA	ANSI	JOD
(	DPS7	DSA	ANSI	JOD
+	DPS7	DSA	ANSI	JOD
*	DPS7	DSA	ANSI	JOD
**	DPS7	DSA	ANSI	JOD
)	DPS7	DSA	ANSI	JOD
;	DPS7	DSA	ANSI	JOD
-	DPS7	DSA	ANSI	JOD
/	DPS7	DSA	ANSI	JOD
,	DPS7	DSA	ANSI	JOD
>	DPS7	DSA	ANSI	JOD
>=	DPS7	DSA	ANSI	JOD
:	DPS7	DSA	ANSI	JOD
=	DPS7	DSA	ANSI	JOD
==	DPS7	DSA	ANSI	JOD
ACCEPT	DPS7	DSA	ANSI	JOD
ACCESS	DPS7	DSA	ANSI	JOD
ACOS	(8)		ANSI	JOD
ACTUAL	DPS7	DSA		
AD				SBS
ADD	DPS7	DSA	ANSI	JOD
ADDRESS	DPS7			
ADVANCING	DPS7	DSA	ANSI	JOD
AFTER	DPS7	DSA	ANSI	JOD
ALIAS	DPS7			SBS
ALL	DPS7	DSA	ANSI	JOD
ALPHABET	DPS7	DSA	ANSI	JOD
ALPHABETIC	DPS7	DSA	ANSI	JOD
ALPHABETIC-LOWER	DPS7	DSA	ANSI	JOD
ALPHABETIC-UPPER	DPS7	DSA	ANSI	JOD
ALPHANUMERIC	DPS7	DSA	ANSI	JOD
ALPHANUMERIC-EDITED	DPS7	DSA	ANSI	JOD
ALSO	DPS7	DSA	ANSI	JOD
ALTER	DPS7	DSA	ANSI	
ALTERING	DPS7			
ALTERNATE	DPS7	DSA	ANSI	JOD
ALTERNATE-CONSOLE	(3)			
ALTERNATE-CONSOLE-X	(7)			
ALTERNATE-CONSOLE-0	(7)			
ALTERNATE-CONSOLE-1	(7)			
ALTERNATE-CONSOLE-2	(7)			
ALTERNATE-CONSOLE-3	(7)			
AND	DPS7	DSA	ANSI	JOD
ANNUITY	(8)		ANSI	JOD
ANSI	(7)			
ANY	DPS7	DSA	ANSI	JOD
APPLY	DPS7			
ARE	DPS7	DSA	ANSI	JOD
AREA	DPS7	DSA	ANSI	JOD

	1	2	3	4
AREAS	DPS7	DSA	ANSI	JOD
ARITHMETIC				JOD
ASA	DPS7			
ASCENDING	DPS7	DSA	ANSI	JOD
ASCII	(1)			
ASIN	(8)		ANSI	JOD
ASSIGN	DPS7	DSA	ANSI	JOD
AT	DPS7	DSA	ANSI	JOD
ATAN	(8)		ANSI	JOD
ATTACH	DPS7			
ATTACH-OPTIONS	DPS7			
AUTHOR	DPS7	DSA	ANSI	
B-AND	DPS7			JOD
B-EXOR	DPS7			JOD
B-LESS				JOD
B-NOT	DPS7			JOD
B-OR	DPS7			JOD
BANNER	(7)			
BECOMES	DPS7			
BEFORE	DPS7	DSA	ANSI	JOD
BEGINNING	DPS7			
BINARY	DPS7	DSA	ANSI	JOD
BIND	(4)			
BIT	DPS7			JOD
BITS	DPS7			JOD
BLANK	DPS7	DSA	ANSI	JOD
BLOCK	DPS7	DSA	ANSI	JOD
BOOLEAN	DPS7			JOD
BOTTOM	DPS7	DSA	ANSI	JOD
BSN	DPS7			
BY	DPS7	DSA	ANSI	JOD
BYTES	(7)			
CALL	DPS7	DSA	ANSI	JOD
CANCEL	DPS7	DSA	ANSI	JOD
CATALOGUE-NAME	DPS7			
CATALOGUED	DPS7			
CD	DPS7	DSA	ANSI	JOD
CF	DPS7	DSA	ANSI	JOD
CH	DPS7	DSA	ANSI	JOD
CHANNEL-1 thru CHANNEL-12	(7)			
CHAR	(8)		ANSI	JOD
CHARACTER	DPS7	DSA	ANSI	JOD
CHARACTERS	DPS7	DSA	ANSI	JOD
CHECK	DPS7			SBS
CHECKPOINT-FILE	DPS7			
CLASS	DPS7	DSA	ANSI	JOD
CLOCK-UNITS	DPS7	DSA	ANSI	
CLOSE	DPS7	DSA	ANSI	JOD
COBOL	DPS7	DSA	ANSI	
CODE	DPS7	DSA	ANSI	JOD
CODE-SET	DPS7	DSA	ANSI	JOD

	1	2	3	4
COLLATING	DPS7	DSA	ANSI	JOD
COLUMN	DPS7	DSA	ANSI	JOD
COMMA	DPS7	DSA	ANSI	JOD
COMMIT	(5)			JOD
COMMON	DPS7	DSA	ANSI	JOD
COMMUNICATION	DPS7	DSA	ANSI	JOD
COMP	DPS7	DSA	ANSI	JOD
COMP-1	DPS7	DSA		JOD
COMP-10	DPS7			JOD
COMP-11	DPS7			JOD
COMP-12	DPS7			JOD
COMP-13	DPS7			JOD
COMP-14	DPS7			JOD
COMP-15	DPS7			JOD
COMP-2	DPS7	DSA		JOD
COMP-3	DPS7	DSA		JOD
COMP-4	DPS7			JOD
COMP-5	DPS7	DSA		JOD
COMP-6	DPS7			JOD
COMP-7	DPS7			JOD
COMP-8	DPS7			JOD
COMP-9	DPS7			JOD
COMPL	DPS7			
COMPLEMENTARY	DPS7			
COMPUTATIONAL	DPS7	DSA	ANSI	JOD
COMPUTATIONAL-1	DPS7	DSA		JOD
COMPUTATIONAL-10	DPS7			JOD
COMPUTATIONAL-11	DPS7			JOD
COMPUTATIONAL-12	DPS7			JOD
COMPUTATIONAL-13	DPS7			JOD
COMPUTATIONAL-14	DPS7			JOD
COMPUTATIONAL-15	DPS7			JOD
COMPUTATIONAL-2	DPS7	DSA		JOD
COMPUTATIONAL-3	DPS7	DSA		JOD
COMPUTATIONAL-4	DPS7			JOD
COMPUTATIONAL-5	DPS7	DSA		JOD
COMPUTATIONAL-6	DPS7			JOD
COMPUTATIONAL-7	DPS7			JOD
COMPUTATIONAL-8	DPS7			JOD
COMPUTATIONAL-9	DPS7			JOD
COMPUTE	DPS7	DSA	ANSI	JOD
CONFIGURATION	DPS7	DSA	ANSI	JOD
CONNECT	(4)			JOD
CONSOLE	(3)	DSA		
CONSOLE-X	(7)			
CONSOLE-0	(7)			
CONSOLE-1	(7)			
CONSOLE-2	(7)			
CONSOLE-3	(7)			
CONSTANT	DPS7			
CONSTRAINT				SBS

	1	2	3	4
CONTAINED				JOD
CONTAINS	DPS7	DSA	ANSI	JOD
CONTENT	DPS7	DSA	ANSI	JOD
CONTINUE	DPS7	DSA	ANSI	JOD
CONTROL	DPS7	DSA	ANSI	JOD
CONTROLS	DPS7	DSA	ANSI	JOD
CONVERSION	DPS7			
CONVERTING	DPS7	DSA	ANSI	JOD
COPY	DPS7	DSA	ANSI	JOD
CORR	DPS7	DSA	ANSI	
CORRESPONDING	DPS7	DSA	ANSI	
COS	(8)		ANSI	JOD
COUNT	DPS7	DSA	ANSI	JOD
CS-BASIC	DPS7			
CS-GENERAL	DPS7			
CURRENCY	DPS7	DSA	ANSI	JOD
CURRENT	(4)			JOD
CURRENT-DATE	(8)		ANSI	JOD
DATA	DPS7	DSA	ANSI	JOD
DATE	DPS7	DSA	ANSI	JOD
DATE-COMPILED	DPS7	DSA	ANSI	
DATE-OF-INTEGER	(8)		ANSI	JOD
DATE-WRITTEN	DPS7	DSA	ANSI	
DAY	DPS7	DSA	ANSI	JOD
DAY-OF-INTEGER	(8)		ANSI	JOD
DAY-OF-WEEK	DPS7	DSA	ANSI	JOD
DB	(4)			JOD
DB-ACCESS-CONTROL-KEY	(4)			JOD
DB-CONFLICT	(4)			
DB-CXT	(4)			
DB-DATA-NAME	(4)			JOD
DB-DESCRIPTIONS	(4)			
DB-DETAILED-STATUS	(4)			
DB-EXCEPTION	(4)			JOD
DB-KEY	DPS7			
DB-KEY-NAME	(4)			
DB-MESSAGE-LENGTH	(4)			
DB-MESSAGE-TEXT	(4)			
DB-PARAMETERS	(4)			
DB-PRIVACY-KEY	(4)			
DB-REALM-NAME	(4)			
DB-RECORD-NAME	(4)			JOD
DB-REGISTERS	(4)			
DB-SET-NAME	(4)			JOD
DB-STATEMENT-CODE	(4)			
DB-STATUS	(4)			JOD
DB-STATUS-CODE	(4)			
DE	DPS7	DSA	ANSI	JOD
DEBUG-CONTENTS	DPS7	DSA	ANSI	
DEBUG-ITEM	DPS7	DSA	ANSI	
DEBUG-LINE	DPS7	DSA	ANSI	

	1	2	3	4
DEBUG-NAME	DPS7	DSA	ANSI	
DEBUG-SUB-1	DPS7	DSA	ANSI	
DEBUG-SUB-2	DPS7	DSA	ANSI	
DEBUG-SUB-3	DPS7	DSA	ANSI	
DEBUGGING	DPS7	DSA	ANSI	JOD
DECIMAL-POINT	DPS7	DSA	ANSI	JOD
DECLARATIVES	DPS7	DSA	ANSI	JOD
DEFAULT	DPS7			JOD
DELETE	DPS7	DSA	ANSI	JOD
DELIMITED	DPS7	DSA	ANSI	JOD
DELIMITER	DPS7	DSA	ANSI	JOD
DEPENDING	DPS7	DSA	ANSI	JOD
DESCENDING	DPS7	DSA	ANSI	JOD
DESTINATION	DPS7	DSA	ANSI	JOD
DETACH	DPS7			
DETAIL	DPS7	DSA	ANSI	JOD
DISABLE	DPS7	DSA	ANSI	JOD
DISCONNECT	(4)			JOD
DISPLAY	DPS7	DSA	ANSI	JOD
DISPLAY-1	DPS7			JOD
DISPLAY-2	DPS7			JOD
DISPLAY-3	DPS7			JOD
DISPLAY-4	DPS7			JOD
DIVIDE	DPS7	DSA	ANSI	JOD
DIVISION	DPS7	DSA	ANSI	JOD
DOWN	DPS7	DSA	ANSI	JOD
DUPLICATE	(4)			JOD
DUPLICATES	DPS7	DSA	ANSI	JOD
DYNAMIC	DPS7	DSA	ANSI	JOD
EBCDIC	(1)			
EGI	DPS7	DSA	ANSI	JOD
ELSE	DPS7	DSA	ANSI	JOD
EMI	DPS7	DSA	ANSI	JOD
EMPTY	(4)			JOD
ENABLE	DPS7	DSA	ANSI	JOD
END	DPS7	DSA	ANSI	JOD
END-ADD	DPS7	DSA	ANSI	JOD
END-CALL	DPS7	DSA	ANSI	JOD
END-COMPUTE	DPS7	DSA	ANSI	JOD
END-DELETE	DPS7	DSA	ANSI	JOD
END-DIVIDE	DPS7	DSA	ANSI	JOD
END-EVALUATE	DPS7	DSA	ANSI	JOD
END-IF	DPS7	DSA	ANSI	JOD
END-MULTIPLY	DPS7	DSA	ANSI	JOD
END-OF-PAGE	DPS7	DSA	ANSI	JOD
END-PERFORM	DPS7	DSA	ANSI	JOD
END-READ	DPS7	DSA	ANSI	JOD
END-RECEIVE	DPS7	DSA	ANSI	JOD
END-RETURN	DPS7	DSA	ANSI	JOD
END-REWRITE	DPS7	DSA	ANSI	JOD
END-SEARCH	DPS7	DSA	ANSI	JOD

	1	2	3	4
END-START	DPS7	DSA	ANSI	JOD
END-STRING	DPS7	DSA	ANSI	JOD
END-SUBTRACT	DPS7	DSA	ANSI	JOD
END-INSTRING	DPS7	DSA	ANSI	JOD
END-WRITE	DPS7	DSA	ANSI	JOD
ENDING	DPS7			
ENTER	DPS7	DSA	ANSI	
ENVIRONMENT	DPS7	DSA	ANSI	JOD
EOP	DPS7	DSA	ANSI	JOD
EQUAL	DPS7	DSA	ANSI	JOD
EQUALS	DPS7			JOD
ERASE	(4)			JOD
ERROR	DPS7	DSA	ANSI	JOD
ESCAPE	DPS7			
ESI	DPS7	DSA	ANSI	JOD
EVALUATE	DPS7	DSA	ANSI	JOD
EVERY	DPS7	DSA	ANSI	
EXAMINE	DPS7			
EXCEEDS	DPS7			JOD
EXCEPTION	DPS7	DSA	ANSI	JOD
EXCLUSIVE	(4)			JOD
EXIT	DPS7	DSA	ANSI	JOD
EXTEND	DPS7	DSA	ANSI	JOD
EXTERNAL	DPS7	DSA	ANSI	JOD
FACTORIAL	(8)		ANSI	JOD
FALSE	DPS7	DSA	ANSI	JOD
FD	DPS7	DSA	ANSI	JOD
FETCH				JOD
FILE	DPS7	DSA	ANSI	JOD
FILE-CONTROL	DPS7	DSA	ANSI	JOD
FILE-ID	DPS7			
FILES	DPS7			JOD
FILLER	DPS7	DSA	ANSI	JOD
FINAL	DPS7	DSA	ANSI	JOD
FIND	(4)			JOD
FINISH	(4)			JOD
FIRST	DPS7	DSA	ANSI	JOD
FLR	DPS7	DSA		
FOOTING	DPS7	DSA	ANSI	JOD
FOR	DPS7	DSA	ANSI	JOD
FORMAT	DPS7			JOD
FREE	(4)			JOD
FROM	DPS7	DSA	ANSI	JOD
FUNCTION	DPS7		ANSI	JOD
GBCD	(1)			
GCOS	(7)			
GENERATE	DPS7	DSA	ANSI	JOD
GET	(4)			JOD
GIVING	DPS7	DSA	ANSI	JOD
GLOBAL	DPS7	DSA	ANSI	JOD
GO	DPS7	DSA	ANSI	JOD

	1	2	3	4
GREATER	DPS7	DSA	ANSI	JOD
GROUP	DPS7	DSA	ANSI	JOD
H-2000	(7)			
HBCD	(1)			
HEADING	DPS7	DSA	ANSI	JOD
HIGH-VALUE	DPS7	DSA	ANSI	JOD
HIGH-VALUES	DPS7	DSA	ANSI	JOD
I-O	DPS7	DSA	ANSI	JOD
I-O-CONTROL	DPS7	DSA	ANSI	JOD
IBCD	(1)			
IDENTIFICATION	DPS7	DSA	ANSI	JOD
IDS-II	DPS7			
IF	DPS7	DSA	ANSI	JOB
IMPLIED	(7)			
IN	DPS7	DSA	ANSI	JOD
INCLUDING	(4)			
INDEX	DPS7	DSA	ANSI	JOD
INDEX-1	DPS7			JOD
INDEX-2	DPS7			JOD
INDEXED	DPS7	DSA	ANSI	JOD
INDEXED-EXT	(4)			
INDICATE	DPS7	DSA	ANSI	JOD
INITIAL	DPS7	DSA	ANSI	JOD
INITIALIZE	DPS7	DSA	ANSI	JOD
INITIATE	DPS7	DSA	ANSI	JOD
INPUT	DPS7	DSA	ANSI	JOD
INPUT-OUTPUT	DPS7	DSA	ANSI	JOD
INSPECT	DPS7	DSA	ANSI	JOD
INSTALLATION	DPS7	DSA	ANSI	
INTEGER	(8)		ANSI	JOD
INTEGER-OF-DATE	(8)		ANSI	JOD
INTEGER-OF-DAY	(8)		ANSI	JOD
INTEGER-PART	(8)		ANSI	JOD
INTO	DPS7	DSA	ANSI	JOD
INVALID	DPS7	DSA	ANSI	JOD
INVOKING	DPS7			
IS	DPS7	DSA	ANSI	JOD
JIS	(1)			
JUST	DPS7	DSA	ANSI	JOD
JUSTIFIED	DPS7	DSA	ANSI	JOD
KEEP	(4)			JOD
KEY	DPS7	DSA	ANSI	JOD
KEY-LOCATION	DPS7			
KEYED	DPS7			
KEYS	(4)			
LABEL	DPS7	DSA	ANSI	
LAST	DPS7	DSA	ANSI	JOD
LD	(4)			JOD
LEADING	DPS7	DSA	ANSI	JOD
LEFT	DPS7	DSA	ANSI	SBS
LENGTH	DPS7	DSA	ANSI	JOD

	1	2	3	4
LESS	DPS7	DSA	ANSI	JOD
LEVEL-64	(7)			
LIMIT	DPS7	DSA	ANSI	JOD
LIMITS	DPS7	DSA	ANSI	JOD
LINAGE	DPS7	DSA	ANSI	JOD
LINAGE-COUNTER	DPS7	DSA	ANSI	JOD
LINE	DPS7	DSA	ANSI	JOD
LINE-COUNTER	DPS7	DSA	ANSI	JOD
LINES	DPS7	DSA	ANSI	JOD
LINES-PER-PAGE	DPS7			
LINKAGE	DPS7	DSA	ANSI	JOD
LN-1 thru LN-255	(7)			
LN1 thru LN255	(7)			
LOCALLY				JOD
LOCK	DPS7	DSA	ANSI	JOD
LOCKS	DPS7			
LOG	(8)		ANSI	JOD
LOG10	(8)		ANSI	JOD
LOW-VALUE	DPS7	DSA	ANSI	JOD
LOW-VALUES	DPS7	DSA	ANSI	JOD
LOWER-CASE	(8)		ANSI	JOD
MAPPING				SBS
MAX	(8)		ANSI	JOD
MAXIMUM	(7)			
MEAN	(8)		ANSI	JOD
MEDIAN	(8)		ANSI	JOD
MEMBER	(4)			JOD
MEMBERS	(4)			
MEMBERSHIP	(4)			
MEMORY	DPS7	DSA	ANSI	
MERGE	DPS7	DSA	ANSI	JOD
MESSAGE	DPS7	DSA	ANSI	JOD
MIDRANGE	(8)		ANSI	JOD
MIN	(8)		ANSI	JOD
MINIMUM-DB-KEY	(4)			
MOD	(8)		ANSI	JOD
MODE	DPS7	DSA	ANSI	JOD
MODIFY	(4)	DSA		JOD
MODULES	DPS7	DSA	ANSI	
MONITORED	(4)			
MOVE	DPS7	DSA	ANSI	JOD
MULTIPLE	DPS7	DSA	ANSI	JOD
MULITPLY	DPS7	DSA	ANSI	JOD
NATIVE	DPS7	DSA	ANSI	JOD
NEGATIVE	DPS7	DSA	ANSI	JOD
NEXT	DPS7	DSA	ANSI	JOD
NO	DPS7	DSA	ANSI	JOD
NO-RESIDENT-INDEX	(7)			
NO-SORTED-INDEX	(7)			
NONE				JOD
NOT	DPS7	DSA	ANSI	JOD

	1	2	3	4
NULL	(2)			JOD
NUMBER	DPS7	DSA	ANSI	JOD
NUMBER-OF-PAGES	(4)			
NUMERIC	DPS7	DSA	ANSI	JOD
NUMERIC-EDITED	DPS7	DSA	ANSI	JOD
NUMVAL	(8)		ANSI	JOD
NUMVAL-C	(8)		ANSI	JOD
OBJECT	DPS7			
OBJECT-COMPUTER	DPS7	DSA	ANSI	JOD
OBJECT-PROGRAM	DPS7			
OCCURS	DPS7	DSA	ANSI	JOD
OF	DPS7	DSA	ANSI	JOD
OFF	DPS7	DSA	ANSI	JOD
OMITTED	DPS7	DSA	ANSI	
ON	DPS7	DSA	ANSI	JOD
ONLY	(4)			JOD
OPEN	DPS7	DSA	ANSI	JOD
OPERATIONAL	DPS7			
OPTIONAL	DPS7	DSA	ANSI	JOD
OR	DPS7	DSA	ANSI	JOD
ORD	(8)		ANSI	JOD
ORD-MAX	(8)		ANSI	JOD
ORD-MIN	(8)		ANSI	JOD
ORDER	DPS7	DSA	ANSI	JOD
ORGANIZATION	DPS7	DSA	ANSI	JOD
OTHER	DPS7	DSA	ANSI	JOD
OUTPUT	DPS7	DSA	ANSI	JOD
OVERFLOW	DPS7	DSA	ANSI	JOD
OVERRIDING	(7)			
OWNER	(4)			JOD
PACKED-DECIMAL	DPS7	DSA	ANSI	JOD
PADDING	DPS7	DSA	ANSI	JOD
PAGE	DPS7	DSA	ANSI	JOD
PAGE-COUNTER	DPS7	DSA	ANSI	JOD
PERFORM	DPS7	DSA	ANSI	JOD
PERMANENT	DPS7			
PF	DPS7	DSA	ANSI	JOD
PH	DPS7	DSA	ANSI	JOD
PIC	DPS7	DSA	ANSI	JOD
PICTURE	DPS7	DSA	ANSI	JOD
PLUS	DPS7	DSA	ANSI	JOD
POINTER	DPS7	DSA	ANSI	JOD
POSITION	DPS7	DSA	ANSI	JOD
POSITIVE	DPS7	DSA	ANSI	JOD
PREATTACHED	DPS7			
PRESENT				JOD
PRESENT-VALUE	(8)		ANSI	JOD
PREVIOUS	(7)			
PRIMARY	DPS7			
PRINTING	DPS7	DSA	ANSI	JOD
PRIOR	(4)			JOD

	1	2	3	4
PROCEDURE	DPS7	DSA	ANSI	JOD
PROCEDURES	DPS7	DSA	ANSI	
PROCEED	DPS7	DSA	ANSI	
PROCESS-AREA	DPS7			
PROGRAM	DPS7	DSA	ANSI	JOD
PROGRAM-ID	DPS7	DSA	ANSI	JOD
PROTECTED	(4)			JOD
PURGE	DPS7	DSA	ANSI	JOD
QUEUE	DPS7	DSA	ANSI	JOD
QUEUED	(7)			
QUOTE	DPS7	DSA	ANSI	JOD
QUOTES	DPS7	DSA	ANSI	JOD
RANDOM	DPS7	DSA	ANSI	JOD
RANGE	(8)		ANSI	JOD
RD	DPS7	DSA	ANSI	JOD
READ	DPS7	DSA	ANSI	JOD
READY	(4)			
REALM	(4)			JOD
REALM-NAME	(4)			
REALMS	(4)			
RECEIVE	DPS7	DSA	ANSI	JOD
RECONNECT	(4)			JOD
RECORD	DPS7	DSA	ANSI	JOD
RECORD-NAME	(4)			JOD
RECORDS	DPS7	DSA	ANSI	JOD
REDEFINES	DPS7	DSA	ANSI	JOD
REEL	DPS7	DSA	ANSI	JOD
REFERENCE	DPS7	DSA	ANSI	JOD
REFERENCES	DPS7	DSA	ANSI	
RELATION				JOD
RELATIVE	DPS7	DSA	ANSI	JOD
RELEASE	DPS7	DSA	ANSI	JOD
REM	(8)		ANSI	JOD
REMAINDER	DPS7	DSA	ANSI	JOD
REMOVAL	DPS7	DSA	ANSI	JOD
RENAMES	DPS7	DSA	ANSI	JOD
REPEATED				JOD
REPLACE	DPS7	DSA	ANSI	JOD
REPLACING	DPS7	DSA	ANSI	JOD
REPORT	DPS7	DSA	ANSI	JOD
REPORTING	DPS7	DSA	ANSI	JOD
REPORTS	DPS7	DSA	ANSI	JOD
RERUN	DPS7	DSA	ANSI	
RESERVE	DPS7	DSA	ANSI	JOD
RESET	DPS7	DSA	ANSI	JOD
RETAINING	(4)			JOD
RETENTION	DPS7			
RETRIEVAL	(4)			JOD
RETURN	DPS7	DSA	ANSI	JOD
REVERSE	(8)		ANSI	JOD
REVERSED	DPS7	DSA	ANSI	

	1	2	3	4
REWIND	DPS7	DSA	ANSI	JOD
REWRITE	DPS7	DSA	ANSI	JOD
RF	DPS7	DSA	ANSI	JOD
RH	DPS7	DSA	ANSI	JOD
RIGHT	DPS7	DSA	ANSI	JOD
ROLLBACK	(4)			JOD
ROUNDED	DPS7	DSA	ANSI	JOD
RUN	DPS7	DSA	ANSI	JOD
RUN-UNIT	(4)			
SAME	DPS7	DSA	ANSI	JOD
SARF	DPS7			
SD	DPS7	DSA	ANSI	JOD
SEARCH	DPS7	DSA	ANSI	JOD
SECONDARY	DPS7			
SECTION	DPS7	DSA	ANSI	JOD
SECURITY	DPS7	DSA	ANSI	
SEGMENT	DPS7	DSA	ANSI	JOD
SEGMENT-LIMIT	DPS7	DSA	ANSI	JOD
SELECT	DPS7	DSA	ANSI	JOD
SELECTED	(4)			
SELECTION	DPS7			SBS
SELECTIVE	(4)			
SEND	DPS7	DSA	ANSI	JOD
SENTENCE	DPS7	DSA	ANSI	JOD
SEPARATE	DPS7	DSA	ANSI	JOD
SEQUENCE	DPS7	DSA	ANSI	JOD
SEQUENTIAL	DPS7	DSA	ANSI	JOD
SET	DPS7	DSA	ANSI	JOD
SETS	(4)			
SHARED	(4)			JOD
SIGN	DPS7	DSA	ANSI	JOD
SIN	(8)		ANSI	JOD
SIZE	DPS7	DSA	ANSI	JOD
SORT	DPS7	DSA	ANSI	JOD
SORT-MERGE	DPS7	DSA	ANSI	JOD
SOURCE	DPS7	DSA	ANSI	JOD
SOURCE-COMPUTER	DPS7	DSA	ANSI	JOD
SPACE	DPS7	DSA	ANSI	JOD
SPACES	DPS7	DSA	ANSI	JOD
SPECIAL-NAMES	DPS7	DSA	ANSI	JOD
SQRT	(8)		ANSI	JOD
SS				SBS
SSF	DPS7			
STANDARD	DPS7	DSA	ANSI	JOD
STANDARD-DEVIATION	(8)		ANSI	JOD
STANDARD-1	DPS7	DSA	ANSI	JOD
STANDARD-2	DPS7	DSA	ANSI	JOD
START	DPS7	DSA	ANSI	JOD
STATION	DPS7			
STATISTICS	(7)			
STATUS	DPS7	DSA	ANSI	JOD

	1	2	3	4
STOP	DPS7	DSA	ANSI	JOD
STORE	(4)			JOD
STREAM	DPS7			
STRING	DPS7	DSA	ANSI	JOD
STRUCTURAL				SBS
STRUCTURE				SBS
SUB-QUEUE-1	DPS7	DSA	ANSI	JOD
SUB-QUEUE-2	DPS7	DSA	ANSI	JOD
SUB-QUEUE-3	DPS7	DSA	ANSI	JOD
SUB-SCHEMA	DPS7	DSA		JOD
SUBSTITUTION	DPS7			
SUBSTRACT	DPS7	DSA	ANSI	JOD
SUM	DPS7	DSA	ANSI	JOD
SUPPRESS	DPS7	DSA	ANSI	JOD
SUSPEND	DPS7			
SWITCH-0 thru SWITCH-31	(3)			
SYMBOLIC	DPS7	DSA	ANSI	JOD
SYNC	DPS7	DSA	ANSI	SBS
SYNCHRONIZED	DPS7	DSA	ANSI	SBS
SYSIN	(3)			
SYSIN-X	(7)			
SYSIN-0	(7)			
SYSIN-1	(7)			
SYSIN-2	(7)			
SYSIN-3	(7)			
SYSOUT	(3)			
SYSOUT-X	(7)			
SYSOUT-0	(7)			
SYSOUT-1	(7)			
SYSOUT-2	(7)			
SYSOUT-3	(7)			
SYSTEM	DPS7			
TABLE	DPS7	DSA	ANSI	JOD
TALLYING	DPS7	DSA	ANSI	JOD
TAN	(8)		ANSI	JOD
TAPE	DPS7	DSA	ANSI	JOD
TEMP	(7)			
TENANT	(4)			JOD
TERMINAL	DPS7	DSA	ANSI	JOD
TERMINAL-X	(7)			
TERMINAL-0	(7)			
TERMINAL-1	(7)			
TERMINAL-2	(7)			
TERMINAL-3	(7)			
TERMINATE	DPS7	DSA	ANSI	JOD
TEST	DPS7	DSA	ANSI	JOD
TEXT	DPS7	DSA	ANSI	JOD
THAN	DPS7	DSA	ANSI	JOD
THEN	DPS7	DSA	ANSI	JOD
THROUGH	DPS7	DSA	ANSI	JOD
THRU	DPS7	DSA	ANSI	JOD

	1	2	3	4
TIME	DPS7	DSA	ANSI	JOD
TIMES	DPS7	DSA	ANSI	JOD
TITLE	DPS7			SBS
TO	DPS7	DSA	ANSI	JOD
TOP	DPS7	DSA	ANSI	JOD
TRAILING	DPS7	DSA	ANSI	JOD
TRANSFORM	DPS7			
TRUE	DPS7	DSA	ANSI	JOD
TYPE	DPS7	DSA	ANSI	JOD
UFF	(7)			
UNBANNERED	(7)			
UNBIND	(4)			
UNEQUAL	DPS7			JOD
UNIT	DPS7	DSA	ANSI	JOD
UNSTRING	DPS7	DSA	ANSI	JOD
UNTIL	DPS7	DSA	ANSI	JOD
UP	DPS7	DSA	ANSI	JOD
UPDATE	(4)			JOD
UPON	DPS7	DSA	ANSI	JOD
UPPER-CASE	(8)		ANSI	JOD
USAGE	DPS7	DSA	ANSI	JOD
USAGE-MODE	(4)			JOD
USE	DPS7	DSA	ANSI	JOD
USING	DPS7	DSA	ANSI	JOD
VALID				JOD
VALIDATE				JOD
VALUE	DPS7	DSA	ANSI	JOD
VALUES	DPS7	DSA	ANSI	JOD
VARIANCE	(8)		ANSI	JOD
VARYING	DPS7	DSA	ANSI	JOD
VIA	DPS7			SBS
VIRTUAL	DPS7			
VLR	DPS7	DSA		
WAIT				JOD
WHEN	DPS7	DSA	ANSI	JOD
WHEN-COMPILED	(8)		ANSI	JOD
WITH	DPS7	DSA	ANSI	JOD
WITHIN	(4)			JOD
WORDS	DPS7	DSA	ANSI	
WORKING-STORAGE	DPS7	DSA	ANSI	JOD
WRITE	DPS7	DSA	ANSI	JOD
ZERO	DPS7	DSA	ANSI	JOD
ZEROES	DPS7	DSA	ANSI	JOD
ZEROS	DPS7	DSA	ANSI	JOD

B. Collating Sequences

The table below shows the EBCDIC character set and its correspondence with the COBOL set. It also gives the correspondence between the various alphabets accepted as standard alphabet by the COBOL compiler. The table is arranged in ascending sequence of the EBCDIC code. It gives the occurrence number of the character in each collating sequence followed by its internal hexadecimal value. In the NATIVE/EBCDIC column, the occurrence number in the collating sequence is the 'symbolic-character' used in non numeric literals. The positions corresponding to low-value and high-value in each alphabet are respectively included in dotted and plain line boxed. In the graphic column the space character is shown by 'b'. Optional graphic symbols appear on the left where 2 symbols are given.

GRAPHIC	NATIVE EBCDIC	STANDARD-1 ASCII	JIS	GBCD
	1 (00)	1 (00)	1 (00)	
	2 (01)	2 (01)	2 (01)	
	3 (02)	3 (02)	3 (02)	
	4 (03)	4 (03)	4 (03)	
	5 (04)	157 (9C)	157 (9C)	
	6 (05)	10 (09)	10 (09)	
	7 (06)	135 (86)	135 (86)	
	8 (07)	128 (7F)	128 (7F)	
	9 (08)	152 (97)	152 (97)	
	10 (09)	142 (8D)	142 (8D)	
	11 (0A)	143 (8E)	143 (8E)	
	12 (0B)	12 (0B)	12 (0B)	
	13 (0C)	13 (0C)	13 (0C)	
	14 (0D)	14 (0D)	14 (0D)	
	15 (0E)	15 (0E)	15 (0E)	
	16 (0F)	16 (0F)	16 (0F)	
	17 (10)	17 (10)	17 (10)	
	18 (11)	18 (11)	18 (11)	
	19 (12)	19 (12)	19 (12)	
	20 (13)	20 (13)	20 (13)	

GRAPHIC	NATIVE EBCDIC	STANDARD-1 ASCII	JIS	GBCD
	21 (14) 22 (15) 23 (16) 24 (17) 25 (18) 26 (19) 27 (1A) 28 (1B) 29 (1C) 30 (1D) 31 (1E) 32 (1F) 33 (20) 34 (21) 35 (22) 36 (23) 37 (24) 38 (25) 39 (26) 40 (27) 41 (28) 42 (29) 43 (2A) 44 (2B) 45 (2C) 46 (2D) 47 (2E) 48 (2F) 49 (30) 50 (31) 51 (32) 52 (33) 53 (34) 54 (35) 55 (36) 56 (37) 57 (38) 58 (39) 59 (3A) 60 (3B)	158 (9D) 134 (85) 9 (08) 136 (87) 25 (18) 26 (19) 147 (92) 144 (8F) 29 (1C) 30 (1D) 31 (1E) 32 (1F) 129 (80) 130 (81) 131 (82) 132 (83) 133 (84) 11 (0A) 24 (17) 28 (1B) 137 (88) 138 (89) 139 (8A) 140 (8B) 141 (8C) 6 (05) 7 (06) 8 (07) 145 (90) 146 (91) 23 (16) 148 (93) 149 (94) 150 (95) 151 (96) 5 (04) 153 (98) 154 (99) 155 (9A) 156 (9B)	158 (9D) 134 (85) 9 (08) 136 (87) 25 (18) 26 (19) 147 (92) 144 (8F) 29 (1C) 30 (1D) 31 (1E) 32 (1F) 129 (80) 130 (81) 131 (82) 132 (83) 133 (84) 11 (0A) 24 (17) 28 (1B) 137 (88) 138 (89) 139 (8A) 140 (8B) 141 (8C) 6 (05) 7 (06) 8 (07) 145 (90) 146 (91) 23 (16) 148 (93) 149 (94) 150 (95) 151 (96) 5 (04) 153 (98) 154 (99) 155 (9A) 156 (9B)	

GRAPHIC	NATIVE EBCDIC	STANDARD-1 ASCII	JIS	GBCD
	61 (3C)	21 (14)	21 (14)	
	62 (3D)	22 (15)	22 (15)	
	63 (3E)	159 (9E)	159 (9E)	
	64 (3F)	27 (1A)	27 (1A)	
b	65 (40)	33 (20)	33 (20)	81 (50)
	66 (41)	161 (A0)	162 (A1)	
	67 (42)	162 (A1)	163 (A2)	
	68 (43)	163 (A2)	164 (A3)	
	69 (44)	164 (A3)	165 (A4)	
	70 (45)	165 (A4)	166 (A5)	
	71 (46)	166 (A5)	167 (A6)	
	72 (47)	167 (A6)	168 (A7)	
	73 (48)	168 (A7)	169 (A8)	
	74 (49)	169 (A8)	170 (A9)	
[o	75 (4A)	92 (5B)	92 (5B)	75 (4A)
.	76 (4B)	47 (2E)	47 (2E)	92 (5B)
<	77 (4C)	61 (3C)	61 (3C)	95 (5E)
(	78 (4D)	41 (28)	41 (28)	94 (5D)
+	79 (4E)	44 (2B)	44 (2B)	177 (B0)
! |	80 (4F)	34 (21)	34 (21)	93 (5C)
&	81 (50)	39 (26)	39 (26)	91 (5A)
	82 (51)	170 (A9)	171 (AA)	
	83 (52)	171 (AA)	172 (AB)	
	84 (53)	172 (AB)	173 (AC)	
	85 (54)	173 (AC)	174 (AD)	
	86 (55)	174 (AD)	175 (AE)	
	87 (56)	175 (AE)	176 (AF)	
	88 (57)	176 (AF)	227 (E2)	
	89 (58)	177 (B0)	177 (B0)	
	90 (59)	178 (B1)	228 (E3)	---------
] !	91 (5A)	94 (5D)	94 (5D)	|192 (BF)|
$	92 (5B)	37 (24)	37 (24)	172 (AB)
*	93 (5C)	43 (2A)	43 (2A)	173 (AC)
)	94 (5D)	42 (29)	42 (29)	174 (AD)
;	95 (5E)	60 (3B)	60 (3B)	175 (AE)
^¬	96 (5F)	95 (5E)	95 (5E)	161 (A0)
-	97 (60)	46 (2D)	46 (2D)	171 (AA)
/	98 (61)	48 (2F)	48 (2F)	178 (B1)
	99 (62)	179 (B2)	229 (E4)	
	100 (63)	180 (B3)	230 (E5)	
	101 (64)	181 (B4)	231 (E6)	
	102 (65)	182 (B5)	232 (E7)	
	103 (66)	183 (B6)		
	104 (67)	184 (B7)		
	105 (68)	185 (B8)		
	106 (69)	186 (B9)		

GRAPHIC	NATIVE EBCDIC	STANDARD-1 ASCII	JIS	GBCD
|	107 (6A)	125 (7C)	125 (7C)	
,	108 (6B)	45 (2C)	45 (2C)	128 (BB)
%	109 (6C)	38 (25)	38 (25)	189 (BC)
-	110 (6D)	96 (5F)	96 (5F)	187 (BA)
>	111 (6E)	63 (3E)	63 (3E)	79 (4E)
?	112 (6F)	48 (3F)	48 (3F)	80 (4F)
	113 (70)	187 (BA)		
	114 (71)	188 (BB)		
	115 (72)	189 (BC)		
	116 (73)	190 (BD)		
	117 (74)	191 (BE)		
	118 (75)	192 (BF)		
	119 (76)	193 (C0)		
	120 (77)	194 (C1)		
	121 (78)	195 (C2)		
'	122 (79)	97 (60)	97 (60)	
:	123 (7A)	59 (3A)	59 (3A)	78 (4D)
#	124 (7B)	36 (23)	36 (23)	76 (4B)
@	125 (7C)	65 (40)	65 (40)	77 (4C)
'	126 (7D)	40 (27)	40 (27)	176 (AF)
=	127 (7E)	62 (3D)	62 (3D)	190 (BD)
"	128 (7F)	35 (22)	35 (22)	191 (BE)
	129 (80)	196 (C3)		
a	130 (81)	98 (61)	178 (B1)	
b	131 (82)	99 (62)	179 (B2)	
c	132 (83)	100 (63)	180 (B3)	
d	133 (84)	101 (64)	181 (B4)	
e	134 (85)	102 (65)	182 (B5)	
f	135 (86)	103 (66)	183 (B6)	
g	136 (87)	104 (67)	184 (B7)	
h	137 (88)	105 (68)	185 (B8)	
i	138 (89)	106 (69)	186 (B9)	
	139 (8A)	197 (C4)	187 (BA)	
	140 (8B)	198 (C5)		
	141 (8C)	199 (C6)	188 (BB)	
	142 (8D)	200 (C7)	189 (BC)	
	143 (8E)	201 (C8)	190 (BD)	
	144 (8F)	202 (C9)	191 (BE)	
	145 (90)	203 (CA)	192 (BF)	
j	146 (91)	107 (6A)	193 (C0)	
k	147 (92)	108 (6B)	194 (C1)	
l	148 (93)	109 (6C)	195 (C2)	
m	149 (94)	110 (6D)	196 (C3)	
n	150 (95)	111 (6E)	197 (C4)	

GRAPHIC	NATIVE EBCDIC	STANDARD-1 ASCII	JIS	GBCD
o	151 (96)	112 (6F)	198 (C5)	
p	152 (97)	113 (70)	199 (C6)	
q	153 (98)	114 (71)	200 (C7)	
r	154 (99)	115 (72)	201 (C8)	
	155 (9A)	204 (CB)	202 (C9)	
	156 (9B)	205 (CC)		
	157 (9C)	206 (CD)		
	158 (9D)	207 (CE)	203 (CA)	
	159 (9E)	208 (CF)	204 (CB)	
	160 (9F)	209 (D0)	205 (CC)	
	161 (A0)	210 (D1)		
~	162 (A1)	127 (7E)	127 (7E)	
s	163 (A2)	116 (73)	206 (CD)	
t	164 (A3)	117 (74)	207 (CE)	
u	165 (A4)	118 (75)	208 (CF)	
v	166 (A5)	119 (76)	209 (D0)	
w	167 (A6)	120 (77)	210 (D1)	
x	168 (A7)	121 (78)	211 (D2)	
y	169 (A8)	122 (79)	212 (D3)	
z	170 (A9)	123 (7A)	213 (D4)	
	171 (AA)	211 (D2)	214 (D5)	
	172 (AB)	212 (D3)		
	173 (AC)	213 (D4)	215 (D6)	
	174 (AD)	214 (D5)	216 (D7)	
	175 (AE)	215 (D6)	217 (D8)	
	176 (AF)	216 (D7)	218 (D9)	
	177 (B0)	217 (D8)		
	178 (B1)	218 (D9)		
	179 (B2)	219 (DA)		
	180 (B3)	220 (DB)		
	181 (B4)	221 (DC)		
	182 (B5)	222 (DD)		
	183 (B6)	223 (DE)		
	184 (B7)	224 (DF)		
	185 (B8)	225 (E0)	225 (E0)	
	186 (B9)	226 (E1)	226 (E1)	
	187 (BA)	227 (E2)	219 (DA)	
	188 (BB)	228 (E3)	220 (DB)	
	189 (BC)	229 (E4)	221 (DC)	
	190 (BD)	230 (E5)	222 (DD)	
	191 (BE)	231 (E6)	223 (DE)	
	192 (BF)	232 (E7)	224 (DF)	

GRAPHIC	NATIVE EBCDIC	STANDARD-1 ASCII	JIS	GBCD
{	193 (C0)	124 (7B)	124 (7B)	
A	194 (C1)	66 (41)	66 (41)	82 (51)
B	195 (C2)	67 (42)	67 (42)	83 (52)
C	196 (C3)	68 (43)	68 (43)	84 (53)
D	197 (C4)	69 (44)	69 (44)	85 (54)
E	198 (C5)	70 (45)	70 (45)	86 (55)
F	199 (C6)	71 (46)	71 (46)	87 (56)
G	200 (C7)	72 (47)	72 (47)	88 (57)
H	201 (C8)	73 (48)	73 (48)	89 (58)
I	202 (C9)	74 (49)	74 (49)	90 (59)
	203 (CA)	233 (E8)	233 (E8)	
	204 (CB)	234 (E9)	234 (E9)	
	205 (CC)	235 (EA)	235 (EA)	
	206 (CD)	236 (EB)	236 (EB)	
	207 (CE)	237 (EC)	237 (EC)	
	208 (CF)	238 (ED)	238 (ED)	
}	209 (D0)	126 (7D)	126 (7D)	
J	210 (D1)	75 (4A)	75 (4A)	162 (A1)
K	211 (D2)	76 (4B)	76 (4B)	163 (A2)
L	212 (D3)	77 (4C)	77 (4C)	164 (A3)
M	213 (D4)	78 (4D)	78 (4D)	165 (A4)
N	219 (D5)	79 (4E)	79 (4E)	166 (A5)
O	215 (D6)	80 (4F)	80 (4F)	167 (A6)
P	216 (D7)	81 (50)	81 (50)	168 (A7)
Q	217 (D8)	82 (51)	82 (51)	169 (A8)
R	218 (D9)	83 (52)	83 (52)	170 (A9)
	219 (DA)	239 (EE)	239 (EE)	
	220 (DB)	240 (EF)	240 (EF)	
	221 (DC)	241 (F0)	241 (F0)	
	222 (DD)	242 (F1)	242 (F1)	
	223 (DE)	243 (F2)	243 (F2)	
	224 (DF)	244 (F3)	244 (F3)	
\	225 (E0)	93 (5C)	93 (5C)	
	226 (E1)	160 (9F)	160 (9F)	
S	227 (E2)	84 (53)	84 (53)	179 (B2)
T	228 (E3)	85 (54)	85 (54)	180 (B3)
U	229 (E4)	86 (55)	86 (55)	181 (B4)
V	230 (E5)	87 (56)	87 (56)	182 (B5)
W	231 (E6)	88 (57)	88 (57)	183 (B6)
X	232 (E7)	89 (58)	89 (58)	184 (B7)

The following shows the graphic collating sequences in the various alphabets.

NATIVE and EBCDIC graphic collating sequences

. < (+ | & ! $ *) ; - / , % _ > ? ` : # @ ' = " 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
{ 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 0 1 2 3 4 5 6 7 8 9

STANDARD-1 and ASCII graphic collating sequences

| " # $ % & ' ( ) * + < - ? 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ 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 ! _ ` 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 { }

JIS graphic collating sequence

| " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ 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 ! - ' { } ~ 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

GBCD graphic collating sequence

0 1 2 3 4 5 6 7 8 9 | # @ : > ? b 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 _ , % = " !

C. The ANSI Flagger

The American National Standard defines the elements of COBOL and the rules for their use. The standard is used by implementors as the reference authority in developing compilers and by users for writing programs in COBOL. The Standard defines required modules and optional modules. Required modules are gathered into three subsets: Minimum, Intermediate, and High. Any program written to conform to the Standard must also conform to one of these subsets.

The ANSI Flagger issues a fatal diagnostic (or an observation message when the LOBSERV parameter of the $CBL JCL statement is used) for each source clause or statement that is not included in the level of compilation specified in the LEVEL parameter of the $CBL JCL statement. In the absence of this parameter, LEVEL = ANSI is assumed.

The following list gives for each COBOL element implemented in DPS 7 COBOL the level(s) to which it pertains, i.e. the option(s) of the 'LEVEL =' parameter of the $CBL JCL statement that should be used when compiling a program containing the element.

Used options have the following meaning:

NSTD	implemented on DPS 7
ANSI	specified in the 1985 COBOL American National Standard (ANS), either in required or optional modules
HIGH	included in the high level of required modules
INTR	included in the intermediate level of required modules
MINI	included in the minimum level of required modules
RPW	included in the Report Writer optional module
1COM	included in the Communication level 1 optional module
2COM	included in the Communication level 2 optional module
1DEB	included in the Debug level 1 optional module
2DEB	included in the Debug level 2 optional module
1SEG	included in the Segmentation level 1 optional module
2SEG	included in the Segmentation level 2 optional module

The letter Z in the right margin indicates that the element is an obsolete element in the Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

LANGUAGE CONCEPTS

```
Character set
Characters used for words
   0,1,...,9, A,B,...,Z - (hyphen or
    minus) . . . . . .  NSTD ANSI HIGH INTR MINI
lower-case accepted
      equivalent to upper-case . . .  NSTD
      not equivalent to upper-case .  NSTD
   _ (underscore) only if the word
    starts with "H_" . . . . . .  NSTD
Characters used for punctuation
   . " ( ) = space . . . . . .  NSTD ANSI HIGH INTR MINI
   ==. . . . . .  NSTD ANSI HIGH
   , ; . . . . . .  NSTD ANSI HIGH INTR MINI
   ' (apostrophe). . . . . .  NSTD
   : (colon) . . . . . .  NSTD ANSI HIGH
   horizontal tabulation . . . . . .  NSTD
Characters used in arithmetic
 operations
   + - * / **. . . . . .  NSTD ANSI HIGH
Characters used in relations
   > < = >= <= . . . . . .  NSTD ANSI HIGH INTR MINI
Characters used in editing
   B 0 + - CR DB Z * $ , . / . . . .  NSTD ANSI HIGH INTR MINI
   E . . . . . .  NSTD
Characters used in subscripting + -.  NSTD ANSI HIGH INTR MINI
Single character substitution allowed NSTD ANSI HIGH INTR MINI
Double character substitution allowed NSTD ANSI HIGH INTR MINI
Separators
" ( ) . , ; space. . . . . .  NSTD ANSI HIGH INTR MINI
' (apostrophe) . . . . . .  NSTD
horizontal tabulation. . . . . .  NSTD
: (colon). . . . . .  NSTD ANSI HIGH
Character-strings
COBOL words
   not more than 30 characters . . .  NSTD ANSI HIGH INTR MINI
   underscore accepted in a name
    provided it begins with "H_"; a
    name may end with underscore . .  NSTD
   User-defined words
      alphabet-name. . . . . .  NSTD ANSI HIGH INTR MINI
      cd-name. . . . . .  NSTD ANSI           1COM
      class-name . . . . . .  NSTD ANSI HIGH INTR MINI
      condition-name . . . . . .  NSTD ANSI HIGH
      data-name
         begins with a letter. . . .  NSTD ANSI HIGH INTR MINI
         begins with a digit . . . .  NSTD ANSI HIGH INTR MINI
      file-name. . . . . .  NSTD ANSI HIGH INTR MINI
      index-name . . . . . .  NSTD ANSI HIGH INTR MINI
      level-number . . . . . .  NSTD ANSI HIGH INTR MINI
      library-name . . . . . .  NSTD ANSI HIGH
      mnemonic-name. . . . . .  NSTD ANSI HIGH INTR MINI
      paragraph-name . . . . . .  NSTD ANSI HIGH INTR MINI
      program-name . . . . . .  NSTD ANSI HIGH INTR MINI
      record-name. . . . . .  NSTD ANSI HIGH INTR MINI
      report-name. . . . . .  NSTD ANSI           RPW
      routine-name . . . . . .  NSTD ANSI HIGH INTR MINI Z
      section-name . . . . . .  NSTD ANSI HIGH INTR MINI
      segment-number . . . . . .  NSTD ANSI           1SEG Z
         symbolic-character. . . . .  NSTD ANSI HIGH
         text-name . . . . . .  NSTD ANSI HIGH INTR
```

```
System-names
      computer-name. . . . . .  NSTD ANSI HIGH INTR MINI
      implementor-name . . . . .  NSTD ANSI HIGH INTR MINI
      system-names are reserved words NSTD ANSI HIGH INTR MINI
      language-name. . . . . .  NSTD ANSI HIGH INTR MINI Z
Reserved words
      Key words. . . . . .  NSTD ANSI HIGH INTR MINI
      Special-character words
         arithmetic operators. . .  NSTD ANSI HIGH
         + or - in subscripting. . .  NSTD ANSI HIGH INTR MINI
         relation characters . . . .  NSTD ANSI HIGH INTR MINI
      Optional words . . . . . .  NSTD ANSI HIGH INTR MINI
      Connectives
         qualifier connectives: OF,
          IN . . . . . .  NSTD ANSI HIGH
         series connectives: ,
          (separator comma ) and ;
          (separator semi-colon) . .  NSTD ANSI HIGH
         logical connectives: AND,
          OR, AND NOT, OR NOT. . . .  NSTD ANSI HIGH
      Special registers
         LINE-COUNTER, PAGE-COUNTER.  NSTD ANSI           RPW
         LINAGE-COUNTER. . . . . .  NSTD ANSI HIGH
         DEBUG-ITEM. . . . . .  NSTD ANSI           1DEB Z
        TALLY. . . . . .  NSTD
        LENGTH OF data-name. . . . .  NSTD
     Figurative constants
        ZERO . . . . . .  NSTD ANSI HIGH INTR MINI
        ZEROS, ZEROES. . . . . .  NSTD ANSI HIGH INTR MINI
        SPACE. . . . . .  NSTD ANSI HIGH INTR MINI
        SPACES . . . . . .  NSTD ANSI HIGH INTR MINI
        HIGH-VALUE, LOW-VALUE. . . .  NSTD ANSI HIGH INTR MINI
        HIGH-VALUES, LOW-VALUES. . .  NSTD ANSI HIGH INTR MINI
        QUOTE. . . . . .  NSTD ANSI HIGH INTR MINI
        QUOTES . . . . . .  NSTD ANSI HIGH INTR MINI
        Symbolic Character . . . . .  NSTD ANSI HIGH
        ALL. . . . . .  NSTD ANSI HIGH
     I-O status. . . . . .  NSTD ANSI HIGH INTR MINI
Literals
   non-numeric literals have lengths
    from 1 through 160 characters. .  NSTD ANSI HIGH INTR MINI
   non-numeric literals have lengths
    from 1 through 256 characters. .  NSTD
   non-numeric literals may contain
    embedded symbolic characters . .  NSTD
      space not required after commas NSTD
   apostrophe accepted as non-numeric
    literal delimiter (for the entire
    program) . . . . . .  NSTD
   numeric literals have lengths from
    1 through 18 digits. . . . . .  NSTD ANSI HIGH INTR MINI
   numeric literals have lengths from
    1 through 30 digits. . . . . .  NSTD
   floating-point numeric literal. .  NSTD
Picture character-strings. . . . . .  NSTD ANSI HIGH INTR MINI
Comment-entries. . . . . .  NSTD ANSI HIGH INTR MINI Z
Qualification
no qualification permitted . . . . .  NSTD ANSI HIGH INTR MINI
50 qualifiers permitted. . . . . .  NSTD ANSI HIGH
```

```
Subscripting
3 levels . . . . . .  NSTD ANSI HIGH INTR MINI
7 levels . . . . . .  NSTD ANSI HIGH
over 7 levels. . . . .  NSTD
subscripting with a literal. . . .  NSTD ANSI HIGH INTR MINI
subscripting with a data-name. . .  NSTD ANSI HIGH INTR MINI
relative subscripting. . . . . .  NSTD ANSI HIGH INTR MINI
subscripting with an expression. . .  NSTD
Reference modification . . . . . .  NSTD ANSI HIGH
   expressions in reference
    modification . . . . . .  NSTD
Reference format
   Sequence numbers. . . . . .  NSTD ANSI HIGH INTR MINI
      special processing of
       horizontal tabulation . . . .  NSTD
   Area A
      Division header. . . . . .  NSTD ANSI HIGH INTR MINI
      Section header . . . . . .  NSTD ANSI HIGH INTR MINI
      Paragraph header . . . . . .  NSTD ANSI HIGH INTR MINI
      Data Division entries. . . . .  NSTD ANSI HIGH INTR MINI
      special processing of
       horizontal tabulation . . . .  NSTD
   Area B
      Paragraphs . . . . . .  NSTD ANSI HIGH INTR MINI
      Data Division entries. . . . .  NSTD ANSI HIGH INTR MINI
   Continuation of lines
      Non-numeric literals . . . . .  NSTD ANSI HIGH INTR MINI
      Words and numeric literals . .  NSTD ANSI HIGH
      PICTURE character-strings. . .  NSTD ANSI HIGH
      special processing of
       horizontal tabulation . . . .  NSTD
      special processing of backspace NSTD
   Blank lines . . . . . .  NSTD ANSI HIGH INTR MINI
   Comment lines
      Asterisk (*) comment lines . .  NSTD ANSI HIGH INTR MINI
      Stroke (/) comment lines . . .  NSTD ANSI HIGH INTR MINI
   Debugging lines with D in indicator
    area . . . . . .  NSTD ANSI HIGH INTR MINI
Source program structure
   Control division optional . . . .  NSTD
   Identification division required.  NSTD ANSI HIGH INTR MINI
   Environment division optional . .  NSTD ANSI HIGH INTR MINI
   Data division optional. . . . . .  NSTD ANSI HIGH INTR MINI
   Procedure division optional . . .  NSTD ANSI HIGH INTR MINI
   End program header. . . . . .  NSTD ANSI HIGH
   Nested source programs
      At the end of the procedure
       division. . . . . .  NSTD ANSI HIGH
      Anywhere an imperative statement
       is allowed. . . . . .  NSTD
```

CONTROL DIVISION.

```
Substitution Section
   The REPLACE clause. . . . . .  NSTD
      LEADING/TRAILING phrase. . .  NSTD
Default Section
   The SYMBOLIC QUEUE clause . . . .  NSTD
   The DISPLAY SIGN clause . . . . .  NSTD
   The COMPUTATIONAL/COMP clause
      DISPLAY. . . . . .  NSTD
      BINARY . . . . . .  NSTD
      PACKED-DECIMAL . . . . . .  NSTD
      COMPUTATIONAL-1/COMP-1 . . . .  NSTD
      COMPUTATIONAL-2/COMP-2 . . . .  NSTD
      COMPUTATIONAL-3/COMP-3 . . . .  NSTD
      COMPUTATIONAL-5/COMP-5 . . . .  NSTD
      COMPUTATIONAL-8/COMP-8 . . . .  NSTD
   The TEMP clause . . . . . .  NSTD
      NOT STANDARD phrase. . . . . .  NSTD
      BINARY phrase. . . . . .  NSTD
   The ACCEPT clause
      SYSIN. . . . . .  NSTD
      CONSOLE. . . . . .  NSTD
      ALTERNATE CONSOLE. . . . . .  NSTD
      TERMINAL . . . . . .  NSTD
   The DISPLAY clause
      SYSOUT . . . . . .  NSTD
      CONSOLE. . . . . .  NSTD
      ALTERNATE CONSOLE. . . . . .  NSTD
      TERMINAL . . . . . .  NSTD
   The SYSIN clause
      SYSIN-0. . . . . .  NSTD
      SYSIN-1. . . . . .  NSTD
      SYSIN-2. . . . . .  NSTD
      SYSIN-X. . . . . .  NSTD
   The ACCEPT ALTERNATE-CONSOLE clause
      ALTERNATE-CONSOLE-0. . . . . .  NSTD
      ALTERNATE-CONSOLE-1. . . . . .  NSTD
      ALTERNATE-CONSOLE-2. . . . . .  NSTD
      ALTERNATE-CONSOLE-X. . . . . .  NSTD
   The ACCEPT CONSOLE clause
      CONSOLE-0. . . . . .  NSTD
      CONSOLE-1. . . . . .  NSTD
      CONSOLE-2. . . . . .  NSTD
      CONSOLE-X. . . . . .  NSTD
   The ACCEPT TERMINAL clause
      TERMINAL-0 . . . . . .  NSTD
      TERMINAL-1 . . . . . .  NSTD
      TERMINAL-2 . . . . . .  NSTD
      TERMINAL-X . . . . . .  NSTD
   The SYSOUT clause
      SYSOUT-0 . . . . . .  NSTD
      SYSOUT-1 . . . . . .  NSTD
      SYSOUT-2 . . . . . .  NSTD
      SYSOUT-X . . . . . .  NSTD
   The DISPLAY ALTERNATE-CONSOLE clause
      ALTERNATE-CONSOLE-0. . . . . .  NSTD
      ALTERNATE-CONSOLE-1. . . . . .  NSTD
      ALTERNATE-CONSOLE-2. . . . . .  NSTD
      ALTERNATE-CONSOLE-X. . . . . .  NSTD
```

```
   The DISPLAY CONSOLE clause
      CONSOLE-0. . . . . .  NSTD
      CONSOLE-1. . . . . .  NSTD
      CONSOLE-2. . . . . .  NSTD
      CONSOLE-X. . . . . .  NSTD
   The DISPLAY TERMINAL clause
      TERMINAL-0 . . . . . .  NSTD
      TERMINAL-1 . . . . . .  NSTD
      TERMINAL-2 . . . . . .  NSTD
      TERMINAL-X . . . . . .  NSTD
   The COBOL 1974 clause
      FOR FILE . . . . . .  NSTD
      FOR COMMUNICATION. . . . . .  NSTD
```

IDENTIFICATION DIVISION.

```
The PROGRAM-ID paragraph . . . . . .  NSTD ANSI HIGH INTR MINI
         Program-name. . . . . .  NSTD ANSI HIGH INTR MINI
         COMMON clause . . . . . .  NSTD ANSI HIGH
         INITIAL clause. . . . . .  NSTD ANSI HIGH
The AUTHOR paragraph . . . . . .  NSTD ANSI HIGH INTR MINI Z
The INSTALLATION paragraph . . . . .  NSTD ANSI HIGH INTR MINI Z
The DATE-WRITTEN paragraph . . . . .  NSTD ANSI HIGH INTR MINI Z
The DATE-COMPILED paragraph. . . . .  NSTD ANSI HIGH           Z
   date updating . . . . . .  NSTD ANSI HIGH           Z
The SECURITY paragraph . . . . . .  NSTD ANSI HIGH INTR MINI Z
```

ENVIRONMENT DIVISION.

```
may be empty . . . . . .  NSTD ANSI HIGH INTR MINI
Configuration Section
   may be absent . . . . . .  NSTD ANSI HIGH INTR MINI
   may be empty. . . . . .  NSTD ANSI HIGH INTR MINI
   The SOURCE-COMPUTER paragraph
     may be absent . . . . . .  NSTD ANSI HIGH INTR MINI
     may be empty. . . . . .  NSTD ANSI HIGH INTR MINI
     HIS-SERIES-60 computer-name . .  NSTD
        GCOS . . . . . .  NSTD
        LEVEL-6. . . . . .  NSTD
        LEVEL-61 . . . . . .  NSTD
        LEVEL-62 . . . . . .  NSTD
        LEVEL-64 . . . . . .  NSTD
        LEVEL-66-ASCII . . . . . .  NSTD
        LEVEL-68 . . . . . .  NSTD
     computer-name
        DPS6 . . . . . .  NSTD ANSI HIGH INTR MINI
        DPS7 . . . . . .  NSTD ANSI HIGH INTR MINI
        DPS8 . . . . . .  NSTD ANSI HIGH INTR MINI
        GCOS . . . . . .  NSTD ANSI HIGH INTR MINI
        LEVEL-6. . . . . .  NSTD ANSI HIGH INTR MINI
        LEVEL-61 . . . . . .  NSTD ANSI HIGH INTR MINI
        LEVEL-62 . . . . . .  NSTD ANSI HIGH INTR MINI
        LEVEL-64 . . . . . .  NSTD ANSI HIGH INTR MINI
        LEVEL-66-ASCII . . . . . .  NSTD ANSI HIGH INTR MINI
        LEVEL-68 . . . . . .  NSTD ANSI HIGH INTR MINI
        any non reserved word. . . .  NSTD ANSI HIGH INTR MINI
```

```
      The MEMORY SIZE clause
         integer WORDS. . . . . .  NSTD
         integer CHARACTERS . . . .  NSTD
         integer MODULES. . . . . .  NSTD
         integer BYTES. . . . . .  NSTD
         ADDRESS range. . . . . .  NSTD
         ADDRESS range series . . .  NSTD
      WITH DEBUGGING MODE phrase. . .  NSTD ANSI HIGH INTR MINI
   The OBJECT-COMPUTER paragraph
      may be absent . . . . . .  NSTD ANSI HIGH INTR MINI
      may be empty. . . . . .  NSTD ANSI HIGH INTR MINI
      HIS-SERIES-60 computer-name . .  NSTD
         GCOS . . . . . .  NSTD
            NATIVE is EBCDIC. . . .  NSTD
         LEVEL-6. . . . . .  NSTD
            NATIVE is EBCDIC. . . .  NSTD
         LEVEL-61 . . . . . .  NSTD
            NATIVE is EBCDIC. . . .  NSTD
         LEVEL-62 . . . . . .  NSTD
            NATIVE is EBCDIC. . . .  NSTD
         LEVEL-64 . . . . . .  NSTD
            NATIVE is EBCDIC. . . .  NSTD
         LEVEL-66-ASCII . . . . . .  NSTD
            NATIVE is EBCDIC. . . .  NSTD
         LEVEL-68 . . . . . .  NSTD
            NATIVE is EBCDIC. . . .  NSTD
      computer-name
         DPS6 . . . . . .  NSTD ANSI HIGH INTR MINI
            NATIVE is EBCDIC. . . .  NSTD ANSI HIGH INTR MINI
         DPS7 . . . . . .  NSTD ANSI HIGH INTR MINI
            NATIVE is EBCDIC. . . .  NSTD ANSI HIGH INTR MINI
         DPS8 . . . . . .  NSTD ANSI HIGH INTR MINI
            NATIVE is EBCDIC. . . .  NSTD ANSI HIGH INTR MINI
         GCOS . . . . . .  NSTD ANSI HIGH INTR MINI
            NATIVE is EBCDIC. . . .  NSTD ANSI HIGH INTR MINI
         LEVEL-6. . . . . .  NSTD ANSI HIGH INTR MINI
            NATIVE is EBCDIC. . . .  NSTD ANSI HIGH INTR MINI
         LEVEL-61 . . . . . .  NSTD ANSI HIGH INTR MINI
            NATIVE is EBCDIC. . . .  NSTD ANSI HIGH INTR MINI
         LEVEL-62 . . . . . .  NSTD ANSI HIGH INTR MINI
            NATIVE is EBCDIC. . . .  NSTD ANSI HIGH INTR MINI
         LEVEL-64 . . . . . .  NSTD ANSI HIGH INTR MINI
            NATIVE is EBCDIC. . . .  NSTD ANSI HIGH INTR MINI
         LEVEL-66-ASCII . . . . . .  NSTD ANSI HIGH INTR MINI
            NATIVE is EBCDIC. . . .  NSTD ANSI HIGH INTR MINI
         LEVEL-68 . . . . . .  NSTD ANSI HIGH INTR MINI
            NATIVE is EBCDIC. . . .  NSTD ANSI HIGH INTR MINI
         any non reserved word. . . .  NSTD ANSI HIGH INTR MINI
            NATIVE is EBCDIC. . . .  NSTD ANSI HIGH INTR MINI
      The MEMORY SIZE clause
         integer WORDS. . . . . .  NSTD ANSI HIGH INTR MINI Z
         integer CHARACTERS . . . .  NSTD ANSI HIGH INTR MINI Z
         integer MODULES. . . . . .  NSTD ANSI HIGH INTR MINI Z
         integer BYTES. . . . . .  NSTD
         ADDRESS range. . . . . .  NSTD
         ADDRESS range series . . .  NSTD
```

```
   The PROGRAM COLLATING SEQUENCE
    clause
      alphabet-name. . . . . .  NSTD ANSI HIGH INTR MINI
      NATIVE . . . . . .  NSTD
      STANDARD-1 . . . . . .  NSTD
      STANDARD-2 . . . . . .  NSTD
      EBCDIC . . . . . .  NSTD
      ASCII. . . . . .  NSTD
      GBCD . . . . . .  NSTD
      JIS. . . . . .  NSTD
   The SEGMENT-LIMIT clause. . . .  NSTD ANSI           2SEG Z
   The MAXIMUM DATA SEGMENT clause  NSTD
   The MAXIMUM PROCEDURE SEGMENT
    clause . . . . . .  NSTD
   The MAXIMUM INITIAL DATA SEGMENT
    clause . . . . . .  NSTD
The SPECIAL-NAMES paragraph
   Implementor-name IS mnemonic-name
      CHANNEL-1 through CHANNEL-12  NSTD ANSI HIGH INTR
      LN1 through LN102. . . . . .  NSTD ANSI HIGH INTR
      LN1 through LN255. . . . . .  NSTD ANSI HIGH INTR
      LN-1 through LN-255. . . . .  NSTD ANSI HIGH INTR
      SWITCH-1 through SWITCH-8. .  NSTD ANSI HIGH INTR MINI
      SWITCH-0 through SWITCH-31 .  NSTD ANSI HIGH INTR MINI
      SYSIN. . . . . .  NSTD ANSI HIGH
      SYSIN-0. . . . . .  NSTD ANSI HIGH
      SYSIN-1. . . . . .  NSTD ANSI HIGH
      SYSIN-2. . . . . .  NSTD ANSI HIGH
      SYSIN-X. . . . . .  NSTD ANSI HIGH
      SYSOUT . . . . . .  NSTD ANSI HIGH
      SYSOUT-0 . . . . . .  NSTD ANSI HIGH
      SYSOUT-1 . . . . . .  NSTD ANSI HIGH
      SYSOUT-2 . . . . . .  NSTD ANSI HIGH
      SYSOUT-X . . . . . .  NSTD ANSI HIGH
      CONSOLE. . . . . .  NSTD ANSI HIGH
      CONSOLE-0. . . . . .  NSTD ANSI HIGH
      CONSOLE-1. . . . . .  NSTD ANSI HIGH
      CONSOLE-2. . . . . .  NSTD ANSI HIGH
      CONSOLE-X. . . . . .  NSTD ANSI HIGH
      ALTERNATE-CONSOLE. . . . . .  NSTD ANSI HIGH
      ALTERNATE-CONSOLE-0. . . . .  NSTD ANSI HIGH
      ALTERNATE-CONSOLE-1. . . . .  NSTD ANSI HIGH
      ALTERNATE-CONSOLE-2. . . . .  NSTD ANSI HIGH
      ALTERNATE-CONSOLE-X. . . . .  NSTD ANSI HIGH
      ALTERNATE CONSOLE. . . . . .  NSTD
      ALTERNATE CONSOLE-0. . . . .  NSTD
      ALTERNATE CONSOLE-1. . . . .  NSTD
      ALTERNATE CONSOLE-2. . . . .  NSTD
      ALTERNATE CONSOLE-X. . . . .  NSTD
      TERMINAL . . . . . .  NSTD ANSI HIGH
      TERMINAL-0 . . . . . .  NSTD ANSI HIGH
      TERMINAL-1 . . . . . .  NSTD ANSI HIGH
      TERMINAL-2 . . . . . .  NSTD ANSI HIGH
      TERMINAL-X . . . . . .  NSTD ANSI HIGH
   ON STATUS . . . . . .  NSTD ANSI HIGH INTR MINI
   OFF STATUS. . . . . .  NSTD ANSI HIGH INTR MINI
   Implementor-name series . . . .  NSTD ANSI HIGH INTR MINI
```

```
The alphabet-name clause
   STANDARD-1 . . . . . .  NSTD ANSI HIGH INTR MINI
   STANDARD-2 . . . . . .  NSTD ANSI HIGH INTR MINI
   NATIVE . . . . . .  NSTD ANSI HIGH INTR MINI
   implementor-name
      EBCDIC. . . . . .  NSTD ANSI HIGH INTR MINI
      ASCII . . . . . .  NSTD ANSI HIGH INTR MINI
      GBCD. . . . . .  NSTD ANSI HIGH INTR MINI
      JIS . . . . . .  NSTD ANSI HIGH INTR MINI
   literal. . . . . .  NSTD ANSI HIGH
The CLASS clause. . . . . .  NSTD ANSI HIGH INTR MINI
The CURRENCY SIGN clause. . . .  NSTD ANSI HIGH INTR MINI
   IS figurative constant except
    symbolic character. . . . .  NSTD
   OBJECT SIGN phrase . . . . .  NSTD
The DECIMAL-POINT clause. . . .  NSTD ANSI HIGH INTR MINI
   COMMA. . . . . .  NSTD ANSI HIGH INTR MINI
   DECIMAL-POINT. . . . . .  NSTD
   OBJECT IS COMMA. . . . . .  NSTD
   OBJECT IS DECIMAL-POINT. . .  NSTD
The SYMBOLIC CHARACTER clause .  NSTD ANSI HIGH
   IN alphabet-name phrase. . .  NSTD ANSI HIGH
   IN NATIVE phrase . . . . . .  NSTD
   IN STANDARD-1 phrase . . . .  NSTD
   IN STANDARD-2 phrase . . . .  NSTD
   IN EBCDIC phrase . . . . . .  NSTD
   IN ASCII phrase. . . . . .  NSTD
   IN GBCD phrase . . . . . .  NSTD
   IN JIS phrase. . . . . .  NSTD
```

Input-Output Section

```
The FILE-CONTROL paragraph
   The SELECT clause. . . . . .  NSTD ANSI HIGH INTR MINI
   The SELECT OPTIONAL clause . .  NSTD ANSI HIGH
   The SELECT EXTERNAL clause . .  NSTD
   The ASSIGN TO implementor-name
    clause
      ifn . . . . . .  NSTD ANSI HIGH INTR MINI
      ifn-CARD-READER . . . . . .  NSTD ANSI HIGH INTR MINI
      ifn-CARD-PUNCH. . . . . .  NSTD ANSI HIGH INTR MINI
      ifn-PRINTER . . . . . .  NSTD ANSI HIGH INTR MINI
      ifn-TAPE. . . . . .  NSTD ANSI HIGH INTR MINI
      ifn-MSD . . . . . .  NSTD ANSI HIGH INTR MINI
      ifn-SYSIN . . . . . .  NSTD ANSI HIGH INTR MINI
      ifn-SYSOUT. . . . . .  NSTD ANSI HIGH INTR MINI
      H-SORT (sort-merge file). .  NSTD ANSI HIGH INTR
      H_SORT (sort-merge file). .  NSTD
      any ifn accepted for sort-merge
       file . . . . . .  NSTD ANSI HIGH INTR
      literal . . . . . .  NSTD ANSI HIGH INTR MINI
```

```
   The ORGANIZATION clause
      SEQUENTIAL. . . . . .  NSTD ANSI HIGH INTR MINI
         default is UFF . . . . .  NSTD ANSI HIGH INTR MINI
         default is ANSI for files
          whose code-set is ASCII or
          STANDARD-1 when the ifn is
          suffixed by -TAPE . . .  NSTD ANSI HIGH INTR MINI
      UFF SEQUENTIAL. . . . . .  NSTD
      QUEUED SEQUENTIAL . . . . .  NSTD
      ANSI SEQUENTIAL . . . . . .  NSTD
      RELATIVE. . . . . .  NSTD ANSI HIGH INTR
         default is UFF . . . . .  NSTD ANSI HIGH INTR
      UFF RELATIVE. . . . . .  NSTD
      INDEXED . . . . . .  NSTD ANSI HIGH INTR
         default is UFF . . . . .  NSTD ANSI HIGH INTR
      UFF INDEXED . . . . . .  NSTD
   The ACCESS MODE clause
      SEQUENTIAL. . . . . .  NSTD ANSI HIGH INTR MINI
      RANDOM. . . . . .  NSTD ANSI HIGH INTR
      DYNAMIC . . . . . .  NSTD ANSI HIGH
      The RELATIVE KEY phrase . .  NSTD ANSI HIGH INTR
   The ACTUAL KEY clause. . . . .  NSTD
   The RECORD KEY clause. . . . .  NSTD ANSI HIGH INTR
   The ALTERNATE RECORD KEY clause NSTD ANSI HIGH
   The FILE STATUS clause . . . .  NSTD ANSI HIGH INTR MINI
   The WITH ASA clause. . . . . .  NSTD
   The WITH SSF clause. . . . . .  NSTD
   The WITH SARF clause . . . . .  NSTD
   The WITH FLR clause. . . . . .  NSTD
   The WITH VLR clause. . . . . .  NSTD
   The WITH BSN clause. . . . . .  NSTD
   The WITH NO BSN clause . . . .  NSTD
   default is BSN . . . . . .  NSTD
   WITH ... is a clause in itself  NSTD
   PADDING CHARACTER clause . . .  NSTD ANSI HIGH
   RECORD DELIMITER clause. . . .  NSTD ANSI HIGH
   RESERVE AREA clause. . . . . .  NSTD ANSI HIGH
The I-O-CONTROL paragraph
   APPLY NO-SORTED-INDEX. . . . .  NSTD
   The MULTIPLE FILE TAPE clause.  NSTD ANSI HIGH           Z
   The RERUN clause . . . . . .  NSTD ANSI HIGH INTR MINI Z
      ON CHECKPOINT-FILE. . . . .  NSTD ANSI HIGH INTR MINI Z
      EVERY integer RECORDS . . .  NSTD ANSI HIGH INTR MINI Z
      EVERY END OF REEL/UNIT. . .  NSTD ANSI HIGH INTR MINI Z
   The SAME AREA clause . . . . .  NSTD ANSI HIGH INTR MINI
   The SAME RECORD AREA clause. .  NSTD ANSI HIGH INTR
   The SAME SORT AREA clause. . .  NSTD ANSI HIGH INTR
   The SAME SORT-MERGE AREA clause NSTD ANSI HIGH INTR
   SAME series. . . . . .  NSTD ANSI HIGH INTR MINI
```

DATA DIVISION.

```
Communication Section. . . . . .  NSTD ANSI           1COM
File Section . . . . . .  NSTD ANSI HIGH INTR MINI
Linkage Section. . . . . .  NSTD ANSI HIGH INTR MINI
Report Section . . . . . .  NSTD ANSI           RPW
Working-Storage Section. . . . . .  NSTD ANSI HIGH INTR MINI
Constant Section . . . . . .  NSTD
The Communication Description entry
   FOR INPUT clause. . . . . .  NSTD ANSI           1COM
      INITIAL phrase . . . . . .  NSTD ANSI           2COM
      SYMBOLIC SUB-QUEUE-X clause. .  NSTD ANSI           2COM
      data-name series . . . . . .  NSTD ANSI           2COM
   FOR OUTPUT clause . . . . . .  NSTD ANSI           1COM
      DESTINATION COUNT clause . . .  NSTD ANSI           1COM
      one or greater . . . . . .  NSTD ANSI           2COM
      DESTINATION TABLE clause . . .  NSTD ANSI           2COM
   FOR I-O clause. . . . . .  NSTD ANSI           1COM
      INITIAL clause . . . . . .  NSTD ANSI           2COM
      data-names series. . . . . .  NSTD ANSI           2COM
   four levels of queue permitted. .  NSTD ANSI           1COM
The Data Description entry . . . . .  NSTD ANSI HIGH INTR MINI
The File Description entry . . . . .  NSTD ANSI HIGH INTR MINI
The Record Description entry . . . .  NSTD ANSI HIGH INTR MINI
The Report Description entry . . . .  NSTD ANSI           RPW
The Report Group Description entry .  NSTD ANSI           RPW
The Sort-Merge Description entry . .  NSTD ANSI HIGH INTR
The BLANK WHEN ZERO clause . . . . .  NSTD ANSI HIGH INTR MINI
The BLOCK CONTAINS clause
   integer CHARACTERS/RECORDS. . . .  NSTD ANSI HIGH INTR MINI
   integer-1 TO integer-2
    CHARACTERS/RECORDS . . . . . .  NSTD ANSI HIGH
The CODE clause. . . . . .  NSTD ANSI           RPW
The CODE-SET clause
   allowed for any file organization  NSTD
   alphabet-name . . . . . .  NSTD ANSI HIGH INTR MINI
   NATIVE. . . . . .  NSTD
   STANDARD-1. . . . . .  NSTD
   STANDARD-2. . . . . .  NSTD
   EBCDIC. . . . . .  NSTD
   ASCII . . . . . .  NSTD
   GBCD. . . . . .  NSTD
   JIS . . . . . .  NSTD
The COLUMN NUMBER clause . . . . . .  NSTD ANSI           RPW
The CONTROL clause . . . . . .  NSTD ANSI           RPW
The data-name clause . . . . . .  NSTD ANSI HIGH INTR MINI
The DATA RECORDS clause. . . . . .  NSTD ANSI HIGH INTR MINI Z
The EXTERNAL clause at level FD or 01 NSTD ANSI HIGH
The EXTERNAL clause at level 77. . .  NSTD
The GLOBAL clause. . . . . .  NSTD ANSI HIGH
   In SD entry . . . . . .  NSTD
   In Communication Section. . . . .  NSTD
   In Linkage Section. . . . . .  NSTD
FILLER . . . . . .  NSTD ANSI HIGH INTR MINI
   may be at any level . . . . . .  NSTD ANSI HIGH INTR MINI
   need not be present . . . . . .  NSTD ANSI HIGH INTR MINI
The GROUP INDICATE clause. . . . . .  NSTD ANSI           RPW
The JUSTIFIED clause (may be abbreviated
 JUST) . . . . . .  NSTD ANSI HIGH INTR MINI
The LABEL RECORDS clause
   STANDARD/OMITTED. . . . . .  NSTD ANSI HIGH INTR MINI Z
Level-number
   1 through 49 (level-number may be 1
```

```
    or 2 digits) . . . . . .  NSTD ANSI HIGH INTR MINI
   66 or 88. . . . . .  NSTD ANSI HIGH
   77. . . . . .  NSTD ANSI HIGH INTR MINI
The LINAGE clause. . . . . .  NSTD ANSI HIGH
The LINE NUMBER clause . . . . . .  NSTD ANSI           RPW
The NEXT GROUP clause. . . . . .  NSTD ANSI           RPW
The OCCURS clause
   integer TIMES . . . . . .  NSTD ANSI HIGH INTR MINI
   ASCENDING/DESCENDING data-name. .  NSTD ANSI HIGH
   ASCENDING/DESCENDING data-name
    series NSTD ANSI HIGH INDEXED BY
     index-name. . . . . .  NSTD ANSI HIGH INTR MINI
   integer-1 TO integer-2 DEPENDING ON
    data-name. . . . . .  NSTD ANSI HIGH
   integer-1 may be zero . . . . . .  NSTD ANSI HIGH
The PAGE clause. . . . . .  NSTD ANSI           RPW
The PICTURE clause (may be abbreviated
 PIC)
   character-string may contain 30
    characters . . . . . .  NSTD ANSI HIGH INTR MINI
   data characters: A X 9. . . . . .  NSTD ANSI HIGH INTR MINI
   operational symbols: S V P. . . .  NSTD ANSI HIGH INTR MINI
   fixed insertion characters:
      0 B , . $ + - DB CR /. . . . .  NSTD ANSI HIGH INTR MINI
   replacement or floating characters:
      + - Z $ *. . . . . .  NSTD ANSI HIGH INTR MINI
   currency sign substitution. . . .  NSTD ANSI HIGH INTR MINI
   decimal point substitution. . . .  NSTD ANSI HIGH INTR MINI
   L with DEPENDING ON . . . . . .  NSTD
   floating point symbol: E. . . . .  NSTD
The RECORD clause. . . . . .  NSTD ANSI HIGH INTR MINI
   VARYING IN SIZE phrase. . . . . .  NSTD ANSI HIGH
   FROM integer TO integer CHARACTERS NSTD ANSI HIGH
   DEPENDING ON. . . . . .  NSTD ANSI HIGH
The REDEFINES clause
   may not be nested . . . . . .  NSTD ANSI HIGH INTR MINI
   may be nested . . . . . .  NSTD ANSI HIGH
   re-definition may be shorter at
    non 01 level . . . . . .  NSTD ANSI HIGH INTR MINI
The RENAMES clause . . . . . .  NSTD ANSI HIGH
The REPORT clause. . . . . .  NSTD ANSI           RPW
The SIGN clause. . . . . .  NSTD ANSI HIGH INTR MINI
The SOURCE clause. . . . . .  NSTD ANSI           RPW
The SUM clause . . . . . .  NSTD ANSI           RPW
The SYNCHRONIZED clause (may be
 abbreviated SYNC) . . . . . .  NSTD ANSI HIGH INTR MINI
The TYPE clause. . . . . .  NSTD ANSI           RPW
```

```
The USAGE clause
   BINARY. . . . . .  NSTD ANSI HIGH INTR MINI
   COMPUTATIONAL (may be abbreviated
    COMP). . . . . .  NSTD ANSI HIGH INTR MINI
      default is COMP-3. . . . . .  NSTD ANSI HIGH INTR MINI
   COMPUTATIONAL-1 (may be abbreviated
    COMP-1). . . . . .  NSTD
   COMPUTATIONAL-2 (may be abbreviated
    COMP-2). . . . . .  NSTD
   COMPUTATIONAL-3 (may be abbreviated
    COMP-3). . . . . .  NSTD
      unsigned is without sign . . .  NSTD
   COMPUTATIONAL-5 (may be abbreviated
    COMP-5). . . . . .  NSTD
   COMPUTATIONAL-8 (may be abbreviated
    COMP-8). . . . . .  NSTD
      byte aligned, unsigned is with
       plus sign . . . . . .  NSTD
   COMPUTATIONAL-9 (may be abbreviated
    COMP-9). . . . . .  NSTD
   COMPUTATIONAL-10 (may be abbreviated
    COMP-10) . . . . . .  NSTD
   COMPUTATIONAL-15 (may be abbreviated
    COMP-15) . . . . . .  NSTD
   DISPLAY . . . . . .  NSTD ANSI HIGH INTR MINI
   INDEX . . . . . .  NSTD ANSI HIGH INTR MINI
   PACKED-DECIMAL. . . . . .  NSTD ANSI HIGH INTR MINI
   POINTER . . . . . .  NSTD
The VALUE clause
   literal . . . . . .  NSTD ANSI HIGH INTR MINI
   literal series. . . . . .  NSTD ANSI HIGH
   literal THRU literal. . . . . .  NSTD ANSI HIGH
   literal range series. . . . . .  NSTD ANSI HIGH
   NULL. . . . . .  NSTD
The VALUE OF clause
   implementor-name IS literal . . .  NSTD ANSI HIGH INTR MINI Z
   implementor-name IS data-name . .  NSTD ANSI HIGH
```

PROCEDURE DIVISION.

```
USING phrase in Procedure Division
 header. . . . . .  NSTD ANSI HIGH INTR MINI
   USING more than 5 data-names. . .  NSTD ANSI HIGH
Declaratives . . . . . .  NSTD ANSI HIGH INTR MINI
Arithmetic expressions . . . . . .  NSTD ANSI HIGH
Conditional expressions
   Simple conditions
      Relation condition
         relation operators
            [NOT] GREATER THAN . . .  NSTD ANSI HIGH INTR MINI
            [NOT] >. . . . . .  NSTD ANSI HIGH INTR MINI
            [NOT] LESS THAN. . . . .  NSTD ANSI HIGH INTR MINI
            [NOT] <. . . . . .  NSTD ANSI HIGH INTR MINI
            [NOT] EQUAL TO . . . . .  NSTD ANSI HIGH INTR MINI
            [NOT] =. . . . . .  NSTD ANSI HIGH INTR MINI
            GREATER THAN OR EQUAL TO  NSTD ANSI HIGH INTR MINI
            LESS THAN OR EQUAL TO. .  NSTD ANSI HIGH INTR MINI
            >= . . . . . .  NSTD ANSI HIGH INTR MINI
            <= . . . . . .  NSTD ANSI HIGH INTR MINI
            EXCEEDS. . . . . .  NSTD
            EQUALS . . . . . .  NSTD
            UNEQUAL TO . . . . . .  NSTD
         boolean operators: B-AND B-OR
          B-EXOR . . . . . .  NSTD
         comparison
            numeric operands . . . .  NSTD ANSI HIGH INTR MINI
            non-numeric operands
               operands must be of equal
                size . . . . . .  NSTD ANSI HIGH INTR MINI
               operands may be unequal
                in size. . . . . .  NSTD ANSI HIGH INTR MINI
            index-names and/or index
             data items. . . . . .  NSTD ANSI HIGH INTR MINI
            boolean operands . . . .  NSTD
            pointers . . . . . .  NSTD
      Class condition (may have a NOT
       option) . . . . . .  NSTD ANSI HIGH INTR MINI
      Switch-Status condition. . . .  NSTD ANSI HIGH INTR MINI
      Condition-Name condition . . .  NSTD ANSI HIGH
      Sign condition (may have a NOT
       option) . . . . . .  NSTD ANSI HIGH
   Complex conditions
      logical operators AND, OR, NOT  NSTD ANSI HIGH
      Negated simple conditions. . .  NSTD ANSI HIGH
      Parenthesized conditions . . .  NSTD ANSI HIGH INTR MINI
      Combined and negated combined
       conditions. . . . . .  NSTD ANSI HIGH
   Abbreviated combined relation
    condition. . . . . .  NSTD ANSI HIGH
```

```
The arithmetic statements
   arithmetic operands limited to 18
    digits . . . . . .  NSTD ANSI HIGH INTR MINI
   arithmetic operands limited to 30
    digits . . . . . .  NSTD
   composite limited to 18 digits. .  NSTD ANSI HIGH INTR MINI
   composite limited to 30 digits. .  NSTD
   intermediate results limited to 18
    digits . . . . . .  NSTD ANSI HIGH INTR MINI
   intermediate results limited to 30
    digits . . . . . .  NSTD
Overlapping operands . . . . . .  NSTD ANSI HIGH INTR MINI
Multiple results in arithmetic
 statements. . . . . .  NSTD ANSI HIGH INTR MINI

The ACCEPT statement
   identifier. . . . . .  NSTD ANSI HIGH INTR MINI
   only one transfer of data . . . .  NSTD ANSI HIGH INTR MINI
   no restriction on the number of
    transfers of data. . . . . .  NSTD ANSI HIGH
   FROM phrase
      mnemonic-name. . . . . .  NSTD ANSI HIGH
      SYSIN. . . . . .  NSTD
      CONSOLE. . . . . .  NSTD
      ALTERNATE CONSOLE. . . . . .  NSTD
      TERMINAL . . . . . .  NSTD
      file-name. . . . . .  NSTD
   default media is SYSIN. . . . . .  NSTD ANSI HIGH INTR MINI
   DATE/DAY/DAY-OF-WEEK/TIME phrase.  NSTD ANSI HIGH
   MESSAGE COUNT phrase. . . . . .  NSTD ANSI           1COM
The ADD statement
   identifier/literal. . . . . .  NSTD ANSI HIGH INTR MINI
   identifier/literal series . . . .  NSTD ANSI HIGH INTR MINI
   TO identifier . . . . . .  NSTD ANSI HIGH INTR MINI
   TO identifier series. . . . . .  NSTD ANSI HIGH INTR MINI
   GIVING identifier . . . . . .  NSTD ANSI HIGH INTR MINI
   GIVING identifier series. . . . .  NSTD ANSI HIGH INTR MINI
   ROUNDED phrase. . . . . .  NSTD ANSI HIGH INTR MINI
   SIZE ERROR phrase . . . . . .  NSTD ANSI HIGH INTR MINI
   ON SIZE ERROR phrase. . . . . .  NSTD ANSI HIGH INTR MINI
   NOT ON SIZE ERROR phrase. . . . .  NSTD ANSI HIGH INTR MINI
   END-ADD phrase. . . . . .  NSTD ANSI HIGH INTR MINI
   CORRESPONDING phrase. . . . . .  NSTD ANSI HIGH
The ALTER statement
   procedure-name. . . . . .  NSTD ANSI HIGH INTR MINI Z
   procedure-name series . . . . . .  NSTD ANSI HIGH           Z
The ASSIGN statement . . . . . .  NSTD
```

```
The CALL statement
   literal . . . . . .  NSTD ANSI HIGH INTR MINI
   identifier. . . . . .  NSTD ANSI HIGH
   USING phrase. . . . . .  NSTD ANSI HIGH INTR MINI
      identifier . . . . . .  NSTD ANSI HIGH INTR MINI
      literal. . . . . .  NSTD
      more than 5 operands . . . . .  NSTD ANSI HIGH
      may have any level-number. . .  NSTD
      may be subscripted . . . . . .  NSTD
      may be an expression . . . . .  NSTD
      BY REFERENCE phrase. . . . . .  NSTD ANSI HIGH
      BY CONTENT phrase. . . . . .  NSTD ANSI HIGH
   USING ADDRESS OF identifier . . .  NSTD
   ON OVERFLOW phrase. . . . . .  NSTD ANSI HIGH
   ON EXCEPTION phrase . . . . . .  NSTD ANSI HIGH
   NOT ON OVERFLOW phrase. . . . . .  NSTD
   NOT ON EXCEPTION phrase . . . . .  NSTD ANSI HIGH
   END-CALL phrase . . . . . .  NSTD ANSI HIGH INTR MINI
The CANCEL statement . . . . . .  NSTD ANSI HIGH
The CLOSE statement
   single file-name. . . . . .  NSTD ANSI HIGH INTR MINI
   file-name series. . . . . .  NSTD ANSI HIGH INTR MINI
   REEL. . . . . .  NSTD ANSI HIGH INTR MINI
   UNIT. . . . . .  NSTD ANSI HIGH INTR MINI
   NO REWIND . . . . . .  NSTD ANSI HIGH
   FOR REMOVAL . . . . . .  NSTD ANSI HIGH
   LOCK. . . . . .  NSTD ANSI HIGH
The COMPUTE statement
   Arithmetic expression . . . . . .  NSTD ANSI HIGH
   Boolean expression. . . . . .  NSTD
   Identifier series . . . . . .  NSTD ANSI HIGH
   ROUNDED phrase. . . . . .  NSTD ANSI HIGH
   ON SIZE ERROR phrase. . . . . .  NSTD ANSI HIGH
   NOT ON SIZE ERROR phrase. . . . .  NSTD ANSI HIGH
   END-COMPUTE phrase. . . . . .  NSTD ANSI HIGH
The CONTINUE statement . . . . . .  NSTD ANSI HIGH INTR MINI
The DELETE statement . . . . . .  NSTD ANSI HIGH INTR
   INVALID KEY phrase. . . . . .  NSTD ANSI HIGH INTR
   NOT INVALID KEY phrase. . . . . .  NSTD ANSI HIGH INTR
   END-DELETE phrase . . . . . .  NSTD ANSI HIGH INTR
The DISABLE statement
   INPUT . . . . . .  NSTD ANSI           2COM
   INPUT TERMINAL. . . . . .  NSTD ANSI           2COM
   OUTPUT. . . . . .  NSTD ANSI           2COM
   KEY identifier/literal. . . . . .  NSTD ANSI           2COM Z
The DISPLAY statement
   only one transfer of data . . . .  NSTD ANSI HIGH INTR MINI
   no restriction on the number of
    transfers of data. . . . . .  NSTD ANSI HIGH
   WITH CONVERSION phrase. . . . . .  NSTD
   UPON phrase
      mnemonic-name. . . . . .  NSTD ANSI HIGH
      SYSOUT . . . . . .  NSTD
      CONSOLE. . . . . .  NSTD
      ALTERNATE CONSOLE. . . . . .  NSTD
      TERMINAL . . . . . .  NSTD
   default media is SYSOUT . . . . .  NSTD ANSI HIGH INTR MINI
   WITH NO ADVANCING phrase. . . . .  NSTD ANSI HIGH
```

```
The DIVIDE statement
   INTO identifier . . . . . .  NSTD ANSI HIGH INTR MINI
   INTO identifier series. . . .  NSTD ANSI HIGH INTR MINI
   BY identifier/literal . . . .  NSTD ANSI HIGH INTR MINI
   GIVING identifier . . . . . .  NSTD ANSI HIGH INTR MINI
   GIVING identifier series. . .  NSTD ANSI HIGH INTR MINI
   REMAINDER phrase. . . . . .  NSTD ANSI HIGH
   ROUNDED phrase. . . . . .  NSTD ANSI HIGH INTR MINI
   SIZE ERROR phrase . . . . . .  NSTD ANSI HIGH INTR MINI
   NOT SIZE ERROR phrase . . . .  NSTD ANSI HIGH INTR MINI
   END-DIVIDE phrase . . . . . .  NSTD ANSI HIGH INTR MINI
The ENABLE statement
   INPUT . . . . . .  NSTD ANSI           2COM
   INPUT TERMINAL. . . . . .  NSTD ANSI           2COM
   OUTPUT. . . . . .  NSTD ANSI           2COM
   KEY identifier/literal. . . .  NSTD ANSI           2COM Z
The ENTER statement. . . . . .  NSTD ANSI HIGH INTR MINI Z
The EVALUATE statement
   Identifier/literal. . . . . .  NSTD ANSI HIGH
   Arithmetic expression . . . .  NSTD ANSI HIGH
   Boolean expression. . . . . .  NSTD
   Conditional expression. . . .  NSTD ANSI HIGH
   TRUE/FALSE. . . . . .  NSTD ANSI HIGH
   ALSO phrase . . . . . .  NSTD ANSI HIGH
   WHEN phrase . . . . . .  NSTD ANSI HIGH
   WHEN OTHER phrase . . . . . .  NSTD ANSI HIGH
   END-EVALUATE phrase . . . . .  NSTD ANSI HIGH
The EXAMINE statement. . . . . .  NSTD
The EXIT statement . . . . . .  NSTD ANSI HIGH INTR MINI
The EXIT PROGRAM statement . . . .  NSTD ANSI HIGH INTR MINI
The GENERATE statement . . . . . .  NSTD ANSI           RPW
The GO TO statement
   procedure-name is required. . .  NSTD ANSI HIGH INTR MINI
   procedure-name is optional. . .  NSTD ANSI HIGH           Z
   DEPENDING ON phrase . . . . .  NSTD ANSI HIGH INTR MINI
The IF statement
   statements must be imperative
    statements . . . . . .  NSTD ANSI HIGH INTR MINI
   Imperative and/or conditional
    statements . . . . . .  NSTD ANSI HIGH
   nested IF statements. . . . .  NSTD ANSI HIGH INTR MINI
   THEN optional word. . . . . .  NSTD ANSI HIGH INTR MINI
   NEXT SENTENCE phrase. . . . .  NSTD ANSI HIGH INTR MINI
   ELSE. . . . . .  NSTD ANSI HIGH INTR MINI
   END-IF phrase . . . . . .  NSTD ANSI HIGH INTR MINI
The INITIALIZE statement
   Identifier series . . . . . .  NSTD ANSI HIGH
   REPLACING phrase. . . . . .  NSTD ANSI HIGH
   REPLACING series. . . . . .  NSTD ANSI HIGH
The INITIATE statement . . . . . .  NSTD ANSI           RPW
```

```
The INSPECT statement
   only single character data item .  NSTD ANSI HIGH INTR MINI
   multi-character data item . . . .  NSTD ANSI HIGH
   TALLYING phrase . . . . . .  NSTD ANSI HIGH INTR MINI
      BEFORE/AFTER phrase. . . . .  NSTD ANSI HIGH INTR MINI
      BEFORE/AFTER phrase series . .  NSTD ANSI HIGH
   TALLYING phrase series. . . . .  NSTD ANSI HIGH
   REPLACING phrase. . . . . .  NSTD ANSI HIGH INTR MINI
      BEFORE/AFTER phrase. . . . .  NSTD ANSI HIGH INTR MINI
      BEFORE/AFTER phrase series . .  NSTD ANSI HIGH
   REPLACING phrase series . . . . .  NSTD ANSI HIGH
   TALLYING and REPLACING phrases. .  NSTD ANSI HIGH INTR MINI
   CONVERTING phrase . . . . . .  NSTD ANSI HIGH
The MERGE statement. . . . . .  NSTD ANSI HIGH INTR
   COLLATING SEQUENCE phrase
      alphabet-name. . . . . .  NSTD ANSI HIGH INTR
      NATIVE . . . . . .  NSTD
      STANDARD-1 . . . . . .  NSTD
      STANDARD-2 . . . . . .  NSTD
      EBCDIC . . . . . .  NSTD
      ASCII. . . . . .  NSTD
      GBCD . . . . . .  NSTD
      JIS. . . . . .  NSTD
The MOVE statement
   TO identifier . . . . . .  NSTD ANSI HIGH INTR MINI
   TO identifier series. . . . . .  NSTD ANSI HIGH INTR MINI
   De-editing. . . . . .  NSTD ANSI HIGH
   CORRESPONDING phrase. . . . . .  NSTD ANSI HIGH
      TO identifier series . . . . .  NSTD
The MULTIPLY statement
   BY identifier . . . . . .  NSTD ANSI HIGH INTR MINI
   BY identifier series. . . . . .  NSTD ANSI HIGH INTR MINI
   GIVING identifier . . . . . .  NSTD ANSI HIGH INTR MINI
   GIVING identifier series. . . . .  NSTD ANSI HIGH INTR MINI
   ROUNDED phrase. . . . . .  NSTD ANSI HIGH INTR MINI
   SIZE ERROR phrase . . . . . .  NSTD ANSI HIGH INTR MINI
   NOT SIZE ERROR phrase . . . . . .  NSTD ANSI HIGH INTR MINI
   END-MULTIPLY phrase . . . . . .  NSTD ANSI HIGH INTR MINI
The OPEN statement
   single file-name. . . . . .  NSTD ANSI HIGH INTR MINI
   file-name series. . . . . .  NSTD ANSI HIGH INTR MINI
   INPUT . . . . . .  NSTD ANSI HIGH INTR MINI
      NO REWIND. . . . . .  NSTD ANSI HIGH
      REVERSED . . . . . .  NSTD ANSI HIGH           Z
   OUTPUT. . . . . .  NSTD ANSI HIGH INTR MINI
      NO REWIND. . . . . .  NSTD ANSI HIGH
   I-O . . . . . .  NSTD ANSI HIGH INTR MINI
   EXTEND. . . . . .  NSTD ANSI HIGH
   INPUT, OUTPUT, I-O, and EXTEND
    series . . . . . .  NSTD ANSI HIGH
   INPUT, OUTPUT, and I-O series . .  NSTD ANSI HIGH INTR MINI
```

```
The PERFORM statement
   procedure-name. . . . . .  NSTD ANSI HIGH INTR MINI
   procedure-name is optional. . . .  NSTD ANSI HIGH INTR MINI
   THRU phrase . . . . . .  NSTD ANSI HIGH INTR MINI
   IMPERATIVE-STATEMENT option . . .  NSTD ANSI HIGH INTR MINI
   END-PERFORM phrase. . . . . .  NSTD ANSI HIGH INTR MINI
   TIMES phrase. . . . . .  NSTD ANSI HIGH INTR MINI
   UNTIL phrase. . . . . .  NSTD ANSI HIGH INTR MINI
       WITH TEST BEFORE/AFTER phrase  NSTD ANSI HIGH
   VARYING phrase. . . . . .  NSTD ANSI HIGH
       WITH TEST BEFORE/AFTER phrase  NSTD ANSI HIGH
   AFTER phrase. . . . . .  NSTD ANSI HIGH
   6 AFTER phrase permitted. . . . .  NSTD ANSI HIGH
The PURGE statement. . . . . .  NSTD ANSI           2COM
The READ statement
   file-name . . . . . .  NSTD ANSI HIGH INTR MINI
   INTO identifier . . . . . .  NSTD ANSI HIGH INTR MINI
   NEXT RECORD . . . . . .  NSTD ANSI HIGH
   PREVIOUS RECORD . . . . . .NSTD
   KEY IS phrase . . . . . .  NSTD ANSI HIGH
   AT END phrase . . . . . .  NSTD ANSI HIGH INTR MINI
   NOT AT END phrase . . . . . .  NSTD ANSI HIGH INTR MINI
   INVALID KEY phrase. . . . . .  NSTD ANSI HIGH INTR
   NOT INVALID KEY phrase. . . . . .  NSTD ANSI HIGH INTR
   END-READ phrase . . . . . .  NSTD ANSI HIGH INTR MINI
The RECEIVE statement
   MESSAGE . . . . . .  NSTD ANSI           1COM
   SEGMENT . . . . . .  NSTD ANSI           2COM
   INTO identifier . . . . . .  NSTD ANSI           1COM
   NO DATA phrase. . . . . .  NSTD ANSI           1COM
   WITH DATA phrase. . . . . .  NSTD ANSI           1COM
   END-RECEIVE phrase. . . . . .  NSTD ANSI           1COM
The RELEASE statement
   record-name . . . . . .  NSTD ANSI HIGH INTR
   FROM phrase . . . . . .  NSTD ANSI HIGH INTR
The RETURN statement
   file-name . . . . . .  NSTD ANSI HIGH INTR
   INTO phrase . . . . . .  NSTD ANSI HIGH INTR
   AT END phrase . . . . . .  NSTD ANSI HIGH INTR
   NOT AT END phrase . . . . . .  NSTD ANSI HIGH INTR
   END-RETURN phrase . . . . . .  NSTD ANSI HIGH INTR
The REWRITE statement
   FROM identifier . . . . . .  NSTD ANSI HIGH INTR MINI
   INVALID KEY phrase. . . . . .  NSTD ANSI HIGH INTR
   NOT INVALID KEY phrase. . . . . .  NSTD ANSI HIGH INTR
   END-REWRITE phrase. . . . . .  NSTD ANSI HIGH INTR MINI
The SEARCH statement . . . . . .  NSTD ANSI HIGH
```

```
The SEND statement
   FROM identifier (portion of a
    message) . . . . . NSTD ANSI          2COM
   FROM identifier (complete message) NSTD ANSI          1COM
   WITH identifier phrase. . . . . NSTD ANSI          2COM
   WITH ESI. . . . . NSTD ANSI          2COM
   WITH EMI. . . . . NSTD ANSI          1COM
   WITH EGI. . . . . NSTD ANSI          1COM
   BEFORE/AFTER ADVANCING
       Integer LINE/LINES. . . . . NSTD ANSI          1COM
       Identifier LINE/LINES . . . . NSTD ANSI          1COM
       Mnemonic-name . . . . . NSTD ANSI          2COM
       PAGE. . . . . NSTD ANSI          1COM
   REPLACING LINE. . . . . NSTD ANSI          2COM
The SET statement
   Index-name/identifier TO. . . . . NSTD ANSI HIGH INTR MINI
   Index-name UP BY/DOWN BY. . . . . NSTD ANSI HIGH INTR MINI
   mnemonic-name . . . . . NSTD ANSI HIGH INTR MINI
   SWITCH-0 through SWITCH-31. . . . NSTD
   TO ON/OFF . . . . . NSTD
   Condition-name TO TRUE. . . . . NSTD ANSI HIGH
   Condition-name TO FALSE . . . . . NSTD
   SET pointer . . . . . NSTD
The SORT statement
   only one SORT statement, a STOP RUN
    statement,  and any associated
    input-output procedures allowed in
    the non-declarative portion of a
    program. . . . . NSTD ANSI HIGH INTR
   program not limited to one SORT
    statement. . . . . NSTD ANSI HIGH INTR
   COLLATING SEQUENCE phrase
      alphabet-name. . . . . NSTD ANSI HIGH INTR
      NATIVE . . . . . NSTD
      STANDARD-1 . . . . . NSTD
      STANDARD-2 . . . . . NSTD
      EBCDIC . . . . . NSTD
      ASCII. . . . . NSTD
      GBCD . . . . . NSTD
      JIS. . . . . NSTD
   WITH DUPLICATES IN SEQUENCE . . . NSTD
The START statement
   EQUAL TO. . . . . NSTD ANSI HIGH
   EQUALS. . . . . NSTD
   = . . . . . NSTD ANSI HIGH
   EXCEEDS . . . . . NSTD
   GREATER . . . . . NSTD ANSI HIGH
   > . . . . . NSTD ANSI HIGH
   LESS . . . . . NSTD
   < . . . . . NSTD
   NOT GREATER . . . . . NSTD
   NOT > . . . . . NSTD
   LESS THAN OR EQUAL TO . . . . . NSTD
   <= . . . . . NSTD
   NOT LESS. . . . . NSTD ANSI HIGH
   NOT < . . . . . NSTD ANSI HIGH
   GREATER THAN OR EQUAL TO. . . . . NSTD ANSI HIGH
   >=. . . . . NSTD ANSI HIGH
   subkey. . . . . NSTD ANSI HIGH
   INVALID KEY phrase. . . . . NSTD ANSI HIGH
   NOT INVALID KEY phrase. . . . . NSTD ANSI HIGH
   END-START phrase. . . . . NSTD ANSI HIGH
```

```
The STOP statement
   RUN . . . . . .  NSTD ANSI HIGH INTR MINI
   literal . . . . . .  NSTD ANSI HIGH INTR MINI Z
   ERROR . . . . . .  NSTD
The STRING statement . . . . . .  NSTD ANSI HIGH
The SUBTRACT statement
   identifier/literal series . . . .  NSTD ANSI HIGH INTR MINI
   FROM identifier . . . . . .  NSTD ANSI HIGH INTR MINI
   FROM identifier series. . . . . .  NSTD ANSI HIGH INTR MINI
   GIVING identifier . . . . . .  NSTD ANSI HIGH INTR MINI
   GIVING identifier series. . . . .  NSTD ANSI HIGH INTR MINI
   ROUNDED phrase. . . . . .  NSTD ANSI HIGH INTR MINI
   SIZE ERROR phrase . . . . . .  NSTD ANSI HIGH INTR MINI
   NOT SIZE ERROR phrase . . . . . .  NSTD ANSI HIGH INTR MINI
   END-SUBTRACT phrase . . . . . .  NSTD ANSI HIGH INTR MINI
   CORRESPONDING phrase. . . . . .  NSTD ANSI HIGH
The SUPPRESS statement . . . . . .  NSTD ANSI           RPW
The TERMINATE statement. . . . . .  NSTD ANSI           RPW
The TRANSFORM statement. . . . . .  NSTD
The UNSTRING statement . . . . . .  NSTD ANSI HIGH
The USE statement
   GLOBAL phrase . . . . . .  NSTD ANSI HIGH
   EXCEPTION/ERROR PROCEDURE
      ON file-name . . . . . .  NSTD ANSI HIGH INTR MINI
      ON file-name series. . . . . .  NSTD ANSI HIGH
      ON INPUT/OUTPUT/I-O. . . . . .  NSTD ANSI HIGH INTR MINI
      ON EXTEND. . . . . .  NSTD ANSI HIGH
   BEFORE REPORTING. . . . . .  NSTD ANSI           RPW
The USE FOR DEBUGGING statement
   procedure-name. . . . . .  NSTD ANSI           1DEB Z
   procedure-name series . . . . . .  NSTD ANSI           1DEB Z
   ALL procedures. . . . . .  NSTD ANSI           1DEB Z
   [ALL REFERENCES OF] identifier
    series . . . . . .  NSTD ANSI           2DEB Z
      WITH CONVERSION. . . . . .  NSTD
   file-name series. . . . . .  NSTD ANSI           2DEB Z
   cd-name series. . . . . .  NSTD ANSI           2DEB Z
The WRITE statement
   record-name . . . . . .  NSTD ANSI HIGH INTR MINI
   FROM identifier . . . . . .  NSTD ANSI HIGH INTR MINI
   BEFORE/AFTER ADVANCING
      integer LINES. . . . . .  NSTD ANSI HIGH INTR MINI
      PAGE . . . . . .  NSTD ANSI HIGH INTR MINI
      identifier LINES . . . . . .  NSTD ANSI HIGH INTR MINI
      mnemonic-name. . . . . .  NSTD ANSI HIGH
   AT END-OF-PAGE phrase . . . . . .  NSTD ANSI HIGH
   INVALID KEY phrase. . . . . .  NSTD ANSI HIGH INTR
   NOT END-OF-PAGE phrase. . . . . .  NSTD ANSI HIGH
   NOT INVALID KEY phrase. . . . . .  NSTD ANSI HIGH INTR
   END-WRITE phrase. . . . . .  NSTD ANSI HIGH INTR MINI
```

Segmentation

Segment-number	NSTD	ANSI	1SEG Z
Fixed segment-number range 0 through 49.	NSTD	ANSI	1SEG Z
Non-fixed segment-number range 50 through 99.	NSTD	ANSI	1SEG Z
Sections with the same segment-number physically contiguous or not. . . .	NSTD	ANSI	1SEG Z
Sections with the same segment-number not physically contiguous are not allowed	NSTD	ANSI	2SEG Z

Library

The COPY statement	NSTD	ANSI	HIGH	INTR
OF/IN library-name.	NSTD	ANSI	HIGH	
REPLACING phrase.	NSTD	ANSI	HIGH	
REPLACING LEADING/TRAILING phrase	NSTD			
The REPLACE statement.	NSTD	ANSI	HIGH	
non pseudo-text	NSTD			

D. The COBOL Obsolete Features

The purpose of the obsolete language element category is to limit the impact of deleting features that are seen as obsolete or improperly specified. Although the elements in this category are obsolete, their abrupt removal from Standard COBOL would be a disservice to COBOL users. Features placed in the obsolete element category have the following characteristics:

- Language elements to be deleted from Standard COBOL will first be identified as obsolete language elements prior to being deleted.
- Obsolete language elements will be neither enhanced, modified, nor maintained.
- The interaction between obsolete language elements and other language elements is undefined unless otherwise specified in Standard COBOL.

The following is a list of the obsolete language elements in third Standard COBOL. Associated with each obsolete element in this list is a justification for placing that element into the obsolete element category.

1. Double character substitution. When a character set contains fewer than 51 characters, double characters must be substituted for the single characters. This feature has been placed in the obsolete element category.

   These specifications are a carry-over from the time when most hardware could not provide the complete COBOL character set.

   This limitation on number of characters available in hardware no longer exists.

2. All literal and numeric or numeric edited form. The figurative constant ALL literal, when associated with a numeric or numeric edited item and when the length of the literal is greater than one, has been placed in the obsolete element category.

   The reason for making this element obsolete is that the results of moving an ALL literal to a numeric data item are often unexpected. For example:

```
01 A PIC 99V99.

    MOVE ALL "99" TO A.
    MOVE ALL "123" TO A.
```

   give values 99.00 and 31.00 respectively.

3. AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED and SECURITY paragraphs. The AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED and SECURITY paragraphs in the Identification Division have been placed in the obsolete element category.

   The purpose of the AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED and SECURITY paragraphs can be achieved through the use of comment lines within the Identification Division since these paragraphs have no effect on the operating of a COBOL program.

   The goal of cleaning up and regularizing the COBOL language has been achieved by declaring many implementor-defined elements as obsolete. The format of the DATE-COMPILED and SECURITY paragraphs are examples of comment-entry paragraphs which are defined by the implementor.

   The interaction of the COPY statement with the comment-entries in the AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED and SECURITY paragraphs is often ambiguous, i.e. the presence of the word COPY in a comment-entry versus the use of the COPY statement in a comment-entry.

4. MEMORY SIZE clause. This MEMORY SIZE clause of the OBJECT-COMPUTER paragraph has been placed in the obsolete element category.

   This anachronistic feature of the language is a carry-over from the time when many systems required a specification of memory size allocation to load the run unit. Memory capacity for a family of main frame models often ranged from 8K to 64K maximum. COBOL programs used the MEMORY SIZE clause to generate objects for specific models.

   This feature is considered to be a function more appropriately controlled by the host operating system in today's computing environment. In second Standard COBOL, the MEMORY SIZE clause was optional. Thus, there are no standard conforming COBOL implementations that require the use of the MEMORY SIZE clause to specify the object computer memory size.

5. RERUN clause. The RERUN clause of the I-O-CONTROL paragraph has been placed in the obsolete element category.

   Seven forms of the RERUN clause are provided. The implementor is required to support at least one form of the RERUN clause.

   This function is considered to be a function more appropriately controlled by the host operating system in today's computing environment.

   The RERUN clause provides only one-half of a complete rerun/restart facility. That is, the syntax and semantics for restart are not specified. Due to the variety in forms of the RERUN clause, there is no guarantee that a program using this clause would be transportable.

6. MULTIPLE FILE TAPE clause. The MULTIPLE FILE TAPE clause in the I-O-CONTROL paragraph of the Environment Division has been placed in the obsolete element category.

   The MULTIPLE FILE TAPE clause should be a function of the operating system and not the individual COBOL program. Therefore, the MULTIPLE FILE TAPE clause has been placed in the obsolete element category.

7. LABEL RECORDS clause. The LABEL RECORDS clause in the file description entry has been placed in the obsolete element category and has been made an optional clause.

   Specifying the presence of file labels is considered a function of the operating system and as such does not belong in the COBOL program.

8. VALUE OF clause. The VALUE OF clause in the file description entry has been placed in the obsolete element category.

   Describing file label items is considered a function of the operating system and does not belong in the COBOL program. Thus the VALUE OF clause has been placed in the obsolete element category.

9. DATA RECORDS clause. The DATA RECORDS clause of the file description entry has been placed in the obsolete element category.

   The DATA RECORDS clause is redundant and may cause misleading documentation.

10. ALTER statement. The ALTER statement has been placed in the obsolete element category.

    The use of the ALTER statement in a program results in a program which may be difficult to understand and maintain. The ALTER statement provides no unique function since the GO TO DEPENDING statement can serve the same purpose.

11. KEY phrase of the DISABLE statement. The KEY phrase of the DISABLE statement has been placed in the obsolete element category and has been made an optional phrase.

    The KEY phrase of the DISABLE statement is used as a password facility for access to the DISABLE statement. However the rules for determining when the value in the KEY phrase matches the system password are not specified, thereby resulting in a situation defined by the implementor. Thus the function provided by the KEY phrase is not portable.

12. KEY phrase of the ENABLE statement. The KEY phrase of the ENABLE statement has been placed in the obsolete element category.

    The KEY phrase of the ENABLE statement is used as a password facility for access to the ENABLE statement. However the rules for determining when the value in the KEY phrase matches the system password are not specified, thereby resulting in a situation defined by the implementor. Thus the function provided by the KEY phrase is not portable.

13. ENTER statement. The ENTER statement has been placed in the obsolete element category.

    The ENTER statement was a precursor of the CALL statement and the calling of external subprograms. The ENTER statement provides no portability because it is optional and is defined by the implementor; thus the ENTER statement is not a good candidate for standardization.

14. The optionality of procedure-name-1 in GO TO statement. The optionality of procedure-name-1 in the GO TO statement has been placed in the obsolete element category.

The optionality of procedure-name-1 in the GO TO statement is dependant upon the ALTER statement. If procedure-name-1 is not specified in format 1 of the GO TO statement, then an ALTER statement referring to that GO TO statement must be executed prior to the execution of the GO TO statement. Since the ALTER statement has been placed in the obsolete element category, the optionality of procedure-name-1 in the GO TO statement has also been placed in the obsolete element category.

15. REVERSED phrase of the OPEN statement. The REVERSED phrase of the OPEN statement has been placed in the obsolete element category.

    A sequential file may be opened for input to be read in reversed order. The necessary hardware to perform this function is not very widely available. Hence, this is an infrequently implemented feature and not a good candidate for standardization. Since this feature is on the hardware dependent list, it is an optional feature which may or may not be implemented.

16. STOP literal statement. The literal variation of the STOP statement has been placed in the obsolete element category.

    General rule 4 of the STOP statement reads: "if STOP literal-1 is specified, the execution of the run unit is suspended and literal-1 is communicated to the operator. Continuation of the execution of the run unit begins with the next executable statement when the implementor-defined procedure governing run unit re-initiation is instituted."

    The function of the STOP literal statement is substantially defined by the implementor and thus not portable.

17. Segmentation module. The segmentation module has been placed in the obsolete element category.

    In the current state of the art, the function provided by the segmentation module is provided at the operating system level, external to the COBOL source code. Thus this feature remains in third Standard COBOL as an obsolete element to be deleted in the next revision.

    Making the segmentation module optional allows existing implementations to continue offering the feature for compatibility reasons, without forcing new implementations to provide a capability grounded in obsolete technology.

18. Debug module. The debug module has been placed in the obsolete element category.

    In the current state of the art, the function provided by the debug module is frequently provided through an interactive debug facility which does not require COBOL source statements. Thus, the feature remains in third Standard COBOL as an obsolete element to be deleted in the next revision.

    Making the debug module optional allows existing implementations to continue offering the feature for compatibility reasons, without forcing new implementations to provide a capability grounded in obsolete technology.

E. COBOL 85 Substantive Changes

E.1 CHANGES NOT AFFECTING EXISTING PROGRAMS

The following is a list of the changes of substance included in third Standard COBOL that are new features not impacting existing programs; for example, a new verb or an additional capability for an old verb.

1. Lower-case letters. Lower-case letters may be used in character-strings. Except when used in non-numeric literals, each is equivalent to the corresponding uppercase letter.

2. Colon (:) character. The COBOL character set has been expanded to include the colon (:) character that is used in reference modification.

3. Punctuation characters. The separators comma, semi-colon and space are interchangeable within a source program.

4. User-defined words and system-names. The same COBOL word may be used as a system-name or as a user-defined word within a source program; the context into which a COBOL word occurs determines what it is.

5. Symbolic-characters. A symbolic-character is a user-defined word that specifies a user-defined figurative constant.

6. Non-numeric literal. A non-numeric literal has an upper limit of 160 characters in length. The upper limit was 120 in second Standard COBOL.

7. Figurative constant ZERO. The figurative constant ZERO is allowed in arithmetic expressions.

8. Uniqueness of reference. A user-defined word need not be unique or be capable of being made unique unless referenced.

9. Qualification. 50 levels of qualification are allowed. Five levels of qualification were possible in second Standard COBOL.

10. Subscripting. A table may have up to seven dimensions. Up to three dimensions were allowed in second Standard COBOL.

11. Relative subscripting. Relative subscripting allows a subscript to be followed by the operator + or - which is followed by an integer.

12. Mixing subscripts and indexes. Indexes and data-name subscripts may both be written in a single set of subscripts used to reference an individual occurrence of a multi-dimensional table.

13. Reference modification. Reference modification is a new method of referencing data by specifying a leftmost character and length for the data item.

14. Sequence number. The sequence number may contain any character of the computer's character set. In second Standard COBOL the sequence number contained only digits.

15. Data Division reference format. The word following a level indicator, level-number 01, or level-number 77 on the same line may begin in area A.

16. End program header. The end program header indicates the end the named COBOL source program; the end program header may be followed by a COBOL program that is to be compiled separately in the same invocation of the compiler.

17. Nested source programs. Programs can be contained in other programs.

18. INITIAL clause in PROGRAM-ID paragraph. The INITIAL clause specifies a program whose state is initialized, whenever the program is called, to the same state as when that program was first called in the run unit.

19. COMMON clause in PROGRAM-ID paragraph. The COMMON clause specifies a program that, despite being directly contained within another program, may be called from any program directly or indirectly contained in that other program.

20. Environment Division. The Environment Division is optional. Within the Environment Division, the Configuration Section is optional. The SOURCE-COMPUTER paragraph, the OBJECT-COMPUTER paragraph, as well as the entries within the SOURCE-COMPUTER paragraph, OBJECT-COMPUTER paragraph, SPECIAL-NAMES paragraph and I-O-CONTROL paragraph are also optional.

21. SPECIAL-NAMES paragraph. Condition-name need not be specified.

22. SPECIAL-NAMES paragraph. The reserved word IS has been made optional in the SPECIAL-NAMES paragraph to be consistent with the use of IS throughout the COBOL specifications.

23. STANDARD-2 option. The STANDARD-2 option within the ALPHABET clause of the SPECIAL-NAMES paragraph allows the specification of ISO 7-bit character set for a character code set or collating sequence.

24. ASSIGN clause. A non-numeric literal may be specified in the ASSIGN clause.

25. OPTIONAL phrase. The OPTIONAL phrase within the file control entry applies to sequential files, relative files, and indexed files opened in the input, I-O, or extend mode. In second Standard COBOL, the OPTIONAL phrase within the file control entry applied to sequential files opened in the input mode.

26. ORGANIZATION clause. Within the ORGANIZATION clause of the file control entry, the words ORGANIZATION IS have been made optional.

27. PADDING CHARACTER clause. The PADDING CHARACTER clause in the file control entry specifies the character which is to be used for block padding on sequential files.

28. RECORD DELIMITER clause. The RECORD DELIMITER clause in the file control entry indicates the method of determining the length of a variable length record on the external medium.

29. I-O-CONTROL paragraph. The order of clauses is immaterial in the I-O-CONTROL paragraph.

30. Data Division. The Data Division is optional.

31. BLOCK CONTAINS clause. Omission of the BLOCK CONTAINS clause is permitted if the number of records contained in a block is specified by the operating environment. In second Standard COBOL, the absence of the BLOCK CONTAINS clause denoted the standard physical record size designated by the implementor.

32. CODE-SET clause. The CODE-SET clause may be specified for all files with sequential organization. In second Standard COBOL, the CODE-SET clause was restricted to non-mass storage files.

33. LABEL RECORDS clause. The LABEL RECORDS clause is optional; if not specified, then the clause LABEL RECORDS ARE STANDARD is assumed.

34. LINAGE clause. Data names within the LINAGE clause may be qualified.

35. EXTERNAL clause. The EXTERNAL clause specifies that a data item or a file connector is external and may be accessed and processed by any program in the run unit.

36. GLOBAL clause. The GLOBAL clause specifies that a data-name or a file-name is a global name that is available to every program contained within the program which declares it.

37. FILLER clause. The use of the word FILLER is optional for data description entries. The word FILLER can appear in a data description entry containing a REDEFINES clause. The word FILLER may be used in a data description entry of a group item.

38. OCCURS clause. The data item specified in the DEPENDING ON phrase may have a zero value. Thus, the minimum number of occurrences may be zero.

39. PICTURE character-string. A PICTURE character-string may be continued between coding lines.

40. PICTURE clause. The insertion character '.' (period) or ',' (comma) may be used as the last character of a PICTURE character-string, provided it is immediately followed by the separator period terminating the data description entry.

41. RECORD clause. The VARYING phrase of the RECORD clause is used to specify variable length records. The DEPENDING phrase associated with the VARYING phrase specifies a data item containing the number of character positions in a record.

42. REDEFINES clause. The size of the item associated with the REDEFINES clause may be less than or equal to the size of the re-defined item. In second Standard COBOL, the two items had to have the same number of character positions.

43. SIGN clause. Multiple SIGN clauses may be specified in the hierarchy of a data description entry; the specification at the subordinate level takes precedence over the specification at the containing group level.

44. SIGN clause. The SIGN clause is allowed in a report group description entry.

45. USAGE clause. BINARY and PACKED-DECIMAL are two new features of the USAGE clause.

46. VALUE clause. The VALUE clause may be specified in a data description entry that contains an OCCURS clause. The VALUE clause may be specified in a data description entry that is subordinate to an entry containing an OCCURS clause. In second Standard COBOL, the VALUE clause was not permitted in a data description entry containing an OCCURS clause or in a data description entry subordinate to an entry containing an OCCURS clause.

47. Communication description entry. The order of clauses in the communication description entry is immaterial.

48. FOR I-O phrase in communication description entry. The FOR I-O phrase in a communication description entry provides for both input and output functions by one CD entry.

49. LINE NUMBER clause. The integer 0 may be specified as the relative line number in the PLUS phrase of the LINE NUMBER clause.

50. Procedure Division. A Linkage Section item which redefines, or is subordinate to one which redefines, an item appearing in the Procedure Division header may be referenced in the Procedure Division.

51. Scope terminators. Scope terminators serve to delimit the scope of certain procedural statements. The scope terminators include: END-ADD, END-CALL, END-COMPUTE, END-DELETE, END-DIVIDE, END-EVALUATE, END-IF, END-MULTIPLY, END-PERFORM, END-READ, END-RECEIVE, END-RETURN, END-REWRITE, END-SEARCH, END-START, END-STRING, END-SUBTRACT, END-UNSTRING, END-WRITE.

52. Relational operators. The relational operator IS GREATER THAN OR EQUAL TO (>=) is equivalent to the relational operator IS NOT LESS THAN. The relational operator IS LESS THAN OR EQUAL TO (<=) is equivalent to the relational operator IS NOT GREATER THAN.

53. Class conditions. Class-name is associated with a set of characters specified by the user in the CLASS clause within the SPECIAL-NAMES paragraph.

54. DAY-OF-WEEK phrase of ACCEPT statement. The DAY-OF-WEEK phrase of the ACCEPT statement provides access to an integer representing the day of week; for example, 1 represents Monday, 2 represents Tuesday, and 7 represent Sunday.

55. ADD statement. The word TO is an optional word in the format: ADD identifier/literal TO identifier/literal GIVING identifier.

56. NOT ON SIZE ERROR phrase of ADD statement. The NOT ON SIZE ERROR phrase provides the programmer with the capability to specify procedures to be executed when a size error condition does not exist for the ADD statement.

57. CALL statement. The BY CONTENT phrase indicates that the called program cannot change the value of a parameter in the CALL statement's USING phrase, but the called program may change the value of the corresponding data item in the called program's Procedure Division header. The BY REFERENCE phrase causes the parameter in the CALL statement's USING phrase to be treated the same as specified in second Standard COBOL.

58. CALL statement. The parameters passed in a CALL statement can be other than 01 or 77 level data item. The parameters passed in a CALL statement may be subscripted and/or reference modified.

59. ON EXCEPTION, NOT ON EXCEPTION phrases of a CALL statement. The ON EXCEPTION phrase of the CALL statement is equivalent to the ON OVERFLOW phrase of the CALL statement. The NOT ON EXCEPTION provides the programmer with the capability to specify procedures to be executed when the program specified by the CALL statement has been made available for execution.

60. REEL/UNIT phrase of the CLOSE statement. The REEL/UNIT phrase of the CLOSE statement can be applied to a single reel/unit file and is specifically permitted for a report file.

61. FOR REMOVAL phrase of the CLOSE statement. The FOR REMOVAL phrase of the CLOSE statement is allowed for a sequential single reel/unit file.

62. NOT ON SIZE ERROR of COMPUTE statement. The NOT ON SIZE ERROR phrase provides the programmer with the capability to specify procedures to be executed when a size error condition does not exist for the COMPUTE statement.

63. CONTINUE statement. The CONTINUE statement indicates that there is no executable statement present and causes an implicit transfer of control to the next executable statement.

64. NOT INVALID KEY phrase of the DELETE statement. The NOT INVALID KEY phrase provides the programmer with the capability to specify procedures to be executed when an invalid key condition does not exist for the DELETE statement.

65. DISPLAY statement. The figurative constant ALL literal is permitted in the DISPLAY statement. In second Standard COBOL, the figurative constant ALL literal was not permitted in the DISPLAY statement.

66. NOT ON SIZE ERROR phrase of DIVIDE statement. The NOT ON SIZE ERROR phrase provides the programmer with the capability to specify procedures to be executed when a size error condition does not exist for the DIVIDE statement.

67. WITH NO ADVANCING phrase of the DISPLAY statement. The WITH NO ADVANCING phrase of the DISPLAY statement provides interaction with a hardware device having vertical positioning.

68. EVALUATE statement. The EVALUATE statement describes a multi-branch, multi-join structure in which multiple conditions are evaluated to determine the subsequent action of the object program.

69. EXIT PROGRAM statement. The EXIT PROGRAM statement need not be the only statement in a paragraph.

70. GO TO DEPENDING statement. The number of procedure-names required in a GO TO DEPENDING statement has been reduced to one.

71. IF statement. The optional word THEN has been added to the general format of the IF statement.

72. INITIALIZE statement. The INITIALIZE statement provides the ability to set selected types of data fields to predetermined values.

73. INSPECT statement. Multiple occurrences of the BEFORE/AFTER phrase allow the TALLYING/REPLACING operation to be initiated after the beginning of the inspection of the data begins and/or terminated before the end of the inspection of the data ends.

74. INSPECT statement. The ALL/LEADING adjective can be distributed over multiple occurrences of the REPLACING CHARACTERS phrase.

75. INSPECT CONVERTING statement. The CONVERTING phrase provides a new variation for the INSPECT statement.

76. MERGE statement. Multiple file-names are allowed in the GIVING phrase of the MERGE statement. A file named in a MERGE statement may contain variable length records. A file named in either the USING or GIVING phrase of a MERGE statement can be a relative file or an indexed file.

77. MOVE statement. A numeric edited data item may be moved to a numeric or numeric edited data item; thus, de-editing takes place.

78. NOT ON SIZE ERROR phrase of the MULTIPLY statement. The NOT ON SIZE ERROR phrase provides the programmer with the capability to specify procedures to be executed when an on size error condition does not exist for the MULTIPLY statement.

79. EXTEND phrase of the OPEN statement. The EXTEND phrase of the OPEN statement can be used with a a relative file or an indexed file.

80. PURGE statement. The PURGE statement causes the message control system (MCS) to eliminate any partial message that has been released by one or more SEND statements.

81. PERFORM statement. Procedure-name may be omitted resulting in an in-line PERFORM of the imperative statement preceding the END-PERFORM phrase terminating the PERFORM statement.

82. PERFORM statement. The TEST AFTER phrase causes the condition to be tested after the specified set of statements has been executed. The TEST BEFORE phrase causes the condition to be tested before the specified set of statements is executed.

83. PERFORM statement. At least six AFTER phrases must be permitted in the VARYING phrase of the PERFORM statement. A maximum of two AFTER phrases existed in second Standard COBOL.

84. READ statement. Variable length records are allowed when the READ statement has an INTO phrase. The NEXT phrase is allowed in a READ statement referencing a file with sequential organization.

85. NOT AT END phrase of READ statement. The NOT AT END phrase provides the programmer with the capability to specify procedures to be executed when the at end condition does not exist for the READ statement.

86. NOT INVALID KEY phrase of READ statement. The NOT INVALID KEY phrase provides the programmer with the capability to specify procedures to be executed when an invalid key condition does not exist for the READ statement.

87. WITH DATA phrase of RECEIVE statement. The WITH DATA phrase provides the programmer with the capability to specify procedures to be executed when the MCS makes data available during execution of a RECEIVE statement.

88. REPLACE statement. The REPLACE statement causes each occurrence of specified text in the source program to be replaced by the corresponding text specified in the REPLACE statement.

89. RETURN statement. Variable length records are allowed when the RETURN statement has an INTO phrase.

90. NOT AT END phrase of the RETURN statement. The NOT AT END phrase provides the programmer with the capability to specify procedures to be executed when an at end condition does not exist for the RETURN statement.

91. REWRITE statement. A record of a different length can replace a record within either a relative or indexed file.

92. NOT INVALID KEY phrase of REWRITE statement. The NOT INVALID KEY phrase provides the programmer with the capability to specify procedures to be executed when an invalid key condition does not exist for the REWRITE statement.

93. SEND statement. The REPLACING LINE phrase is a new feature of the SEND statement.

94. SET statement. Index-names and identifiers may now be mixed in a series of operands preceding the word TO in a SET statement. Two new variations of the SET statement permit the setting of an external switch to be changed and permit the value of a conditional variable to be changed.

95. SORT statement. Multiple file-names are allowed in the GIVING phrase of the SORT statement. A file named in a SORT statement may contain variable length records. A file named in either the USING or GIVING phrase of a SORT statement can be a relative or an indexed file. The files named in the USING and GIVING phrases can reside on the same physical reel. If the DUPLICATES phrase is specified, records whose key values are identical remain in the same order as they were when they were input to the sort process when the sort process is completed.

96. SORT and MERGE statements. The input and output procedures of a SORT or MERGE statement may contain explicit transfers of control to points outside the input or output procedure. The remainder of the Procedure Division may contain transfers of control to points inside the input or output procedure. A paragraph-name may be specified in the INPUT PROCEDURE phrase or the OUTPUT PROCEDURE phrase.

97. NOT INVALID KEY phrase of START statement. The NOT INVALID KEY phrase provides the programmer with the capability to specify procedures to be executed when an invalid key condition does not exist for the START statement.

98. STRING statement. The identifier in the INTO phrase of the STRING statement may be a group item.

99. NOT ON OVERFLOW phrase of STRING statement. The NOT ON OVERFLOW phrase provides the programmer with the capability to specify procedures to be executed when an overflow condition does not exist for the STRING statement.

100. NOT ON SIZE ERROR phrase of the SUBTRACT statement. The NOT ON SIZE ERROR phrase provides the programmer with the capability to specify procedures to be executed when a size error condition does not exist for the SUBTRACT statement.

101. NOT ON OVERFLOW phrase of UNSTRING statement. The NOT ON OVERFLOW phrase provides the programmer with the capability to specify procedures to be executed when an overflow condition does not exist for the UNSTRING statement.

102. USE statement. A USE AFTER EXCEPTION/ERROR declarative statement specifying the name of a file takes precedence over a declarative statement specifying the open mode of the file.

103. USE statement. The GLOBAL phrase specifies that the associated declarative procedures are invoked during the execution of any program contained within the program which includes the USE statement.

104. USE BEFORE REPORTING statement. The GLOBAL phrase specifies that the associated declarative procedures are invoked during the execution of any program contained within the program which includes the USE BEFORE REPORTING statement.

105. NOT-END-OF-PAGE phrase of WRITE statement. The NOT-END-OF-PAGE phrase provides the programmer with the capability to specify procedures to be executed when an end-of-page condition does not exist for the WRITE statement.

106. NOT INVALID KEY phrase of WRITE statement. The NOT INVALID KEY phrase provides the programmer with the capability to specify procedures to be executed when an invalid key condition does not exist for the WRITE statement.

E.2 CHANGES WHICH MAY AFFECT EXISTING PROGRAMS

This section contains a list of the changes of substance included in third Standard COBOL that are new features or changes that could impact existing programs; for example, the addition of a rule for a previously undefined situation or the change of a rule for an existing verb.

The general philosophy in developing third Standard COBOL was that clarifications of unclear or ambiguous rules should be made in the interest of portability of programs and of ease of development of new programs. The addition of new features has also been done with the intent of making new programs easier and lest costly to develop. The changes have been made with the intent of impacting existing programs as little as possible. The long term savings in program portability and development should outweigh the short term costs of conversion of existing programs.

It should be noted that this section contains a list of changes having the potential to impact existing programs. In those cases where second Standard COBOL was unclear, the clarification has been made in accordance with a de facto industry standard, if one existed. In any case, a clarification does not cause an incompatibility between standards; it only causes the possibility of an incompatibility between any particular implementation and third Standard COBOL. The justifications included in the following list address primarily the effects of the changes on COBOL programs which follow the rules of second Standard COBOL. The effects of the changes are not always known for programs that: (1) violate the rules of second Standard COBOL, or (2) use features for which the rules were not well defined in second Standard COBOL and thus where dependent on a particular implementor's extension or interpretation of the rules.

1. Length of ALL literal. When the figurative constant ALL is not associated with another data item, the length of the string is the length of the literal.

   The rules in second Standard COBOL for the size of the figurative constant ALL literal differ depending on where the figurative constant has been used in the program.

2. Alphabet-name clause. The key word ALPHABET must precede alphabet-name within the alphabet-name clause of the SPECIAL-NAMES paragraph.

   In third Standard COBOL, system-names and user-defined words are allowed to intersect. The introduction of the key word ALPHABET in the alphabet-name clause resolves the resulting possible ambiguity.

   This feature makes it easier to move a program from implementation to implementation; system-names no longer need to be changed. To modify an existing program, the key word ALPHABET must be inserted in front of the alphabet-name clause.

3. Collating sequence. The collating sequence used to access an indexed file is the collating sequence associated with the native character set that was in effect for the file at the time the file was created.

   In second Standard COBOL, two interpretations were possible: the native collating sequence, or the collating sequence specified by the PROGRAM COLLATING SEQUENCE clause.

4. CURRENCY SIGN clause. The literal specified within the currency sign clause may not be a figurative constant.

5. RELATIVE KEY phrase. The relative key data item specified in the RELATIVE KEY phrase must not contain the PICTURE symbol 'P'.

6. LINAGE clause. Files for which the LINAGE clause has been specified must not be opened in the extend mode.

7. FOOTING phrase. If the FOOTING phrase is not specified, no end-of-page condition independent of the page overflow condition exists.

   In second Standard COBOL, the specifications for the existence of the footing area are contradictory between the LINAGE clause and the WRITE clause. The solution in third Standard COBOL reflects the intuition that if no footing area is specified, then none is wanted. Thus, if no FOOTING phrase is specified in the LINAGE clause, then no footing area exists and no end-of-page condition occurs.

8. OCCURS clause. When a receiving item is a variable length data item and contains the object of the DEPENDING ON phrase, the maximum length of the item will be used.

   In second Standard COBOL, the length was computed based on the value of the item in the DEPENDING ON phrase prior to the execution of the statement. Using the second Standard COBOL rules with a MOVE statement (or a READ INTO statement) could have resulted in loss of data if the value of the DEPENDING ON data item was not set to indicate the length of the sending data before the MOVE was executed.

   Programs which conform to second Standard COBOL will not be affected by this change in third Standard COBOL.

   To change an existing program which is affected, restructure the affected data records so that there are no valid data items following the transferred part of a variable length data item in a record.

9. PICTURE symbol 'P'. When a data item described by a PICTURE containing the character 'p' is referenced, the digit positions referenced by 'P' will be considered to contain zeroes in the following operations: (1) any operation requiring a numeric sending operand; (2) a MOVE statement where the sending operand is numeric and its PICTURE character-string contains the symbol 'P'; (3) a MOVE statement where the sending operand is numeric edited and its PICTURE character-string contains the symbol 'P' and the receiving operand is numeric or numeric edited; (4) a comparison operation where both operands are numeric.

   In second Standard COBOL, digit positions described with a 'P' where considered to contain zeroes when used in an operation involving conversion of data from one form of internal representation to another. Second Standard COBOL did not specify what happened in operations not involving data conversion, or when conversion was required.

10. Procedure Division header. A data item appearing in the USING phrase of the Procedure Division header must not have a REDEFINES clause in its data description entry.

    In second Standard COBOL, an item which was described with a REDEFINES clause can be specified in the USING phrase of the Procedure Division header. If the calling program specifies two different parameters, the results are undefined.

    In most cases, a program which specified a re-defining item in the USING phrase of the Procedure Division header can be converted by substituting the re-defined item.

11. Exponentiation. The following special cases of exponentiation are defined in third Standard COBOL:

    a. If an expression having a zero value is raised to a negative or zero power, the size error condition exists.

    b. If the evaluation of the exponentiation yields both a positive and a negative number, the positive number is returned.

    c. If no real number exists as the result of the evaluation, the size error condition exists.

    Second Standard COBOL did not state what would happen in these special cases of exponentiation.

12. Order of execution for a conditional expression. Two or more conditions connected by only the logical operator AND or only the logical operator OR within a hierarchical level are evaluated in order from left to right, and evaluation of that hierarchical level terminates as soon as a truth value for it is determined regardless of whether all the constituent connected conditions within that hierarchical level have been evaluated.

    By specifying the order of evaluation, program portability will be enhanced.

13. Class conditions. The ALPHABETIC test is true for uppercase letters, lower-case letters, and the space character. The ALPHABETIC-UPPER test is true for uppercase letters and the space character. The ALPHABETIC-LOWER test is true for lower-case letters and the space character.

    In second Standard COBOL, the ALPHABETIC test was true for uppercase letters and the space character.

    The change from ALPHABETIC to ALPHABETIC-UPPER can be reliably accomplished by an automated source code conversion program.

14. CANCEL statement. The CANCEL closes all open files.

    In second Standard COBOL, the status of files left in the open mode when the program was cancelled is not defined. The change in third Standard COBOL produces a predictable result for processing this statement.

    The only programs that will potentially be affected are those that cancelled programs and expected files associated with the cancelled programs to remain open after the execution of the CANCEL statement.

15. CLOSE statement. The NO REWIND phrase cannot be specified in a CLOSE statement having the REEL/UNIT phrase.

    In second Standard COBOL, the rules for the NO REWIND phrase and the REEL/UNIT phrase were sometimes in conflict.

16. COPY statement. If the word COPY appears in a comment-entry or in the place where a comment-entry may appear, it is considered part of the comment-entry.

    In second Standard COBOL, the appearance of the word COPY in a comment-entry was an undefined situation. The specification of this situation within third Standard COBOL will enhance program portability.

17. COPY statement. After all COPY statements have been processed, a debugging line will be considered to have all the characteristics of a comment line, if the WITH DEBUGGING MODE is not specified in the SOURCE-COMPUTER paragraph.

    Second Standard COBOL did not address the situation of a COPY statement or a portion of a COPY statement appearing on a debugging line.

18. COPY statement. Pseudo-text-1 must not consist entirely of a separator comma or a separator semi-colon.

    Second Standard COBOL allowed pseudo-text-1 to consist entirely of a separator comma or a separator semi-colon but did not specify under what conditions replacement took place. Any attempt to define the semantics in this situation would have caused a potential incompatibility.

19. DISPLAY statement. After the last operand has been transferred to the hardware device, the positioning of the hardware device will be reset to the leftmost position of the next line of the device.

    In second Standard COBOL, the positioning of the hardware device after the last operand was undefined. The new rule in third Standard COBOL is necessary for a complete description of the NO ADVANCING phrase.

20. DIVIDE statement. Any subscripts for identifier-4 in the REMAINDER phrase are evaluated after the result of the DIVIDE operation is stored in identifier-3 of the GIVING phrase.

    In second Standard COBOL, the point at which any subscript in the REMAINDER phrase is determined during the processing of the DIVIDE statement is undefined.

21. EXIT PROGRAM statement. When there is no next executable statement in a called program, an implicit EXIT PROGRAM statement is executed.

    This situation was undefined in second Standard COBOL. Defining it makes programs more transportable.

22. EXIT PROGRAM statement. The following new rule appears for the EXIT PROGRAM statement: "...the ends of the ranges of all PERFORM statements executed by the called program are considered to have been reached."

    This situation is undefined in second Standard COBOL.

23. INSPECT statement. The order of execution for evaluating subscripts in the INSPECT statement is specified. Subscripting associated with any identifier is evaluated only once as the first operation in the execution of the INSPECT statement.

    The order of execution for evaluating subscripts in the INSPECT statement was undefined in second Standard COBOL.

24. MERGE statement. No two files in a MERGE statement may be specified in the SAME AREA or SAME SORT-MERGE AREA clause. The only files in a MERGE statement that can be specified in the SAME RECORD AREA clause are those associated with the GIVING phrase.

    This rule, not present in second Standard COBOL, is a clarification of the interaction of the SAME clause and the MERGE statement. It adds syntactical restrictions against situations which are likely to be troublesome. These situations probably appear in few existing programs.

25. PERFORM statement. The order of initialization of multiple VARYING identifiers in the PERFORM statement is specified.

    The order of initialization of multiple VARYING identifiers in the PERFORM statement was undefined in second Standard COBOL. This change is the resolution of an ambiguity and will promote program portability.

26. PERFORM statement. Within the VARYING...AFTER phrase of the PERFORM statement, identifier-2 is augmented before identifier-5 is set. In second Standard COBOL, identifier-5 was set before identifier-2 was augmented.

    The situation where one VARYING variable depends on another is useful for processing half a matrix along the diagonal; the rules in third Standard COBOL specify this function properly while the rules in second Standard COBOL did not specify this function properly.

27. PERFORM statement. The order of execution for evaluating subscripts in the PERFORM VARYING is specified. This situation was undefined in second Standard COBOL.

    This change is the resolution of an ambiguity and will help promote program portability.

28. READ statement. The INTO phrase cannot be specified: (a) unless all records associated with the file and the data item specified in the INTO phrase are group items or elementary alphanumeric items, or (b) unless only one record description is subordinate to the file description entry.

    In second Standard COBOL, the semantics for the move of the record to the identifier specified in the INTO phrase are not supplied. For a file with multiple elementary records, there is no statement as to whether any conversion of data takes place or whether a group move is performed. The new rules in third Standard COBOL disallow a possible ambiguous situation.

29. RECEIVE statement. If a message size is greater than the area referenced, the message fills the area referenced left to right starting with the leftmost character of the message. Further RECEIVE statements which reference the same queue, sub-queue, ..., must be executed to transfer the remainder of the message in the area referenced.

    In second Standard COBOL, it was not clearly defined whether or not subsequent RECEIVE statements were to be executed in order to receive the remainder of the message.

30. RETURN statement. The INTO phrase cannot be specified: (a) unless all records associated with the file and the data item specified in the INTO phrase are group items or elementary alphanumeric items, or (b) unless only one record description is subordinate to the sort-merge file description entry.

    In second Standard COBOL, the semantics for the move of the record to the identifier specified in the INTO phrase of the RETURN statement are not supplied. For a file with multiple elementary records, there is no statement as to whether any conversion of data takes place or whether a group move is performed.

    Programs affected by this change are those performing a RETURN INTO statement on a file describing multiple elementary records that include at least one numeric record.

31. STOP RUN statement. The STOP RUN statement closes all files.

    In second Standard COBOL, the state of files remaining in the open mode at run completion was not specified. In some cases, this situation could have led to errors.

32. STOP RUN statement. If the run unit has been accessing messages, the STOP RUN statement causes the message control system (MCS) to eliminate from the queue any message partially received by that run unit.

    In second Standard COBOL, it is undefined what happens to partially received messages when a run unit executes a STOP RUN statement.

33. STRING statement. The order of execution for evaluating subscripts in the STRING statement is specified.

    In second Standard COBOL, the order of evaluation of subscripts is not specified; in particular, the relative order of subscript evaluation and pointer modification is undefined.

34. UNSTRING statement. In the UNSTRING statement, any subscripting associated with the DELIMITED BY identifier, the INTO identifier, the DELIMITER IN identifier, or the COUNT IN identifier is evaluated once, immediately before the examination of the sending fields for the delimiter.

35. WRITE statement. The phrases ADVANCING PAGE and END-OF-PAGE must not both be specified in a single WRITE statement.

    In second Standard COBOL, it is possible to specify both of these phrases within one WRITE statement but no rules are provided to identify their order of processing.

36. File position indicator. The concept of a current record pointer in second Standard COBOL has been changed to a file position indicator.

37. File position indicator. For a relative or indexed file in the dynamic access mode, execution of an OPEN I-O statement followed by one or more WRITE statements and then a READ NEXT statement will cause the READ statement to access the first record in the file at the time of execution of the READ statement.

    In second Standard COBOL, this sequence caused the READ statement to access the first record at the time of execution of the OPEN statement. If one of the WRITE statements inserted a record with a key or relative record number lower than that of any records previously existing in the file, a different record would be accessed by the READ statement.

    The semantics in third Standard COBOL bring the situation following an OPEN statement into line with that following a READ statement.

38. File position indicator. If an alternate key is the key of reference and the alternate key is changed by a REWRITE statement to a value between the current value and the next value in the file, a subsequent READ NEXT statement will obtain the same record.
    In second Standard COBOL, the subsequent READ statement would obtain the record with the next value for that alternate key prior to the REWRITE statement.

39. Reserved words. The following reserved words have been added:

ALPHABET	END-DIVIDE	EXTERNAL
ALPHABETIC-LOWER	END-EVALUATE	FALSE
ALPHABETIC-UPPER	END-IF	GLOBAL
ALPHANUMERIC	END-MULTIPLY	INITIALIZE
ALPHANUMERIC-EDITED	END-PERFORM	NUMERIC-EDITED
ANY	END-READ	ORDER
BINARY	END-RECEIVE	OTHER
CLASS	END-RETURN	PACKED-DECIMAL
COMMON	END-REWRITE	PADDING
CONTENT	END-SEARCH	PURGE
CONTINUE	END-START	REFERENCE
CONVERTING	END-STRING	REPLACE
DAY-OF-WEEK	END-SUBTRACT	STANDARD-2
END-ADD	END-UNSTRING	TEST
END-CALL	END-WRITE	THEN
END-COMPUTE	EVALUATE	TRUE
END-DELETE		

In each case, the benefits to be derived from the additional facility provided through the addition of each reserved word were deemed to outweigh the inconvenience caused by removing this word from the realm of user-defined words. It is the intention that the use of the new REPLACE statement will mitigate the inconvenience to existing programs which may use any of the new reserved words as user-defined words.

40. I-O status. New I-O status values have been added.

    Second Standard COBOL specified only a few I-O status code conditions. As a result, the user could not distinguish among many different exceptional conditions which he might wish to treat in a variety of ways, and/or each implementor specified a different set of implementor-defined status codes which covered a variety of situations in a variety of ways. Also, second Standard COBOL left the results of many I-O situations undefined; that is: second Standard COBOL stated that certain criteria were to be met, but not what happened when they were not met; hence, execution of the object program becomes undefined.

    The intention in third Standard COBOL is to define status codes for these undefined I-O situations. Thus the user can check for these error conditions and take corrective action for specific error conditions where appropriate.

    In general, the additions may impact programs:

    a. If they test for specific implementor-defined status values to detect conditions now defined.

    b. If they rely on a successful completion status for any of the conditions now defined. (In the case of new I-O status values 04, 05, and 07, this only affects programs which examine both character positions of the I-O status to check for successful completion.)

    c. If they rely on some implementor-dependent action such as abnormal termination of the program when any of the newly defined conditions arise.

    This change may have a substantial impact on those programs which check specific I-O status values.

    It should be noted that the first Standard COBOL did not provide any status code.

    The individual I-O status values affected are described in the following paragraphs:

    a. I-O status 04. A READ statement is successfully executed but the length of the record processed does not conform to the fixed file attributes for the file.

       Second Standard COBOL does not define the consequence if a READ statement accesses a record containing more or fewer characters than the maximum and minimum, respectively, specified for that file. Thus, the result of reading such a record is undefined. The new I-O status value of 04 alerts the user to this situation.

       Since third Standard COBOL prevents an attempt to write or rewrite a record that is too large or too small, this situation cannot occur for records written by a program on an implementation of third Standard COBOL.

    b. I-O status 05. An OPEN statement is successfully executed but the referenced optional file is not present at the time the OPEN statement is executed.

       According to second Standard COBOL, the absence of an optional file is not signaled to the program until the first READ statement for this file. The new I-O status value of 05 makes the information specific and available at the time the file is referenced by an OPEN statement, allowing the program to take more discerning action with respect to this condition.

c. I-O status 07. The input-output statement is successfully executed. However, for a CLOSE statement with the NO REWIND, REEL/UNIT, or FOR REMOVAL phrase, or for an OPEN statement with the NO REWIND phrase , the referenced file is on a non-reel/unit medium.

According to second Standard COBOL, an OPEN statement with the NO REWIND phrase can only be used with sequential single reel/unit files, and a CLOSE statement with the NO REWIND, REEL/UNIT, or FOR REMOVAL phrase is illegal for a non-reel/unit file. However, with mass storage files, these instances of OPEN or CLOSE can be considered successful in essence, if the anomaly of the NO REWIND, REEL/UNIT, or FOR REMOVAL phrase is overlooked. The new I-O status value of 07 makes successful completion possible, while preserving the information for the user in case he wishes to take specific action.

d. I-O status 14. A sequential READ statement is attempted for a relative file and the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file.

Second Standard COBOL states that successful execution of a format 1 READ statement referencing a relative file updates the content of the relative key data item (if specified) to contain the relative record number of the record made available. Second Standard COBOL does not define the result if the number of significant digits of the relative record number is larger than the relative key data item. The new I-O status value 14 defines the result.

e. I-O status 24. An attempt is made to write beyond the externally defined boundaries of a relative or indexed file; or a sequential WRITE statement is attempted for a relative file and the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file.

In second Standard COBOL, the I-O status value of 24 covers only an attempt to write beyond the externally defined boundaries of a relative or indexed file.

Second Standard COBOL states that on successful execution of a WRITE statement referencing a relative file, the relative record number of the record released will be placed in the relative key data item (if specified). It does not define the result if the number of significant digits of the relative record number is larger than the relative key data item.

Only programs which sequentially write more records than the maximum value allowed by the PICTURE of the relative key data item may be affected by this change.

f. I-O status 35. An OPEN statement with the INPUT phrase is attempted on a non-optional file that is not present.

Second Standard COBOL requires that the OPTIONAL phrase must be specified for input files that are not necessarily present each time the object program is executed. It does not specify what happens when a file which is not declared as optional is absent. The new I-O status value of 35 allows the user to test for this condition.

g. I-O status 37. An OPEN statement is attempted on a file which is required to be a mass storage file but is not.

This new I-O status value will be returned if either: (1) an OPEN I-O statement is attempted for a non-mass storage file, or (2) an OPEN statement is attempted for a non-mass storage file which is declared in the program to be a relative or indexed file.

Second Standard COBOL does not specify what happens in these circumstances. The new I-O status value of 37 permits the user to test for this error condition.

h. I-O status 38. An OPEN statement is attempted on a file previously closed with lock.

Second Standard COBOL specify that a file closed with lock cannot be opened again during the current execution of the run unit, but does not specify what happens if an attempt is made to reopen the file. The new I-O status value of 38 permits the user to test for this condition.

i. I-O status 39. An OPEN statement is unsuccessful because a conflict has been detected between the fixed file attributes and the attributes specified for that file in the program.

Fixed file attributes are attributes of a file which are fixed at the time the file is created and which cannot be changed throughout the lifetime of the file. They are the organization, the code set, the minimum and maximum logical record size, the record type (fixed or variable), the blocking factor, the padding character, and the record delimiter. For indexed files only, additional fixed file attributes are the prime record key, the alternate record keys, and the collating sequence of the keys.

Second Standard COBOL specifies that the file organization is established at the time a file is created and subsequently cannot be changed. It also specifies for an OPEN INPUT, OPEN I-O, or OPEN EXTEND statement that the file description of the file, which includes the CODE-SET, RECORD, and BLOCK CONTAINS clauses must be equivalent to that used when the file was created. The ability to specify a padding character and a record delimiter are new facilities not available in second Standard COBOL. For indexed files, second Standard COBOL specifies that the data descriptions and relative locations within a record of the record key and alternate record key data items, and the number of alternate record keys, must be the same as when the file was created. Second Standard COBOL does not provide the ability to influence the collating sequence used for the keys of an indexed file.

Second Standard COBOL does not specify what happens if the fixed file attributes conflict with the attributes specified for a file in the program. The new I-O status value of 39 allows the user to test for this condition.

j. I-O status 41. An OPEN statement is attempted for a file in the open mode.

Second Standard COBOL does not allow an OPEN statement to refer to a file in open mode, but does not define the consequence of such a reference. The new I-O status value of 41 permits the user to test for the condition.

k. I-O status 42. A CLOSE statement is attempted for a file not in the open mode.

Second Standard COBOL does not allow a CLOSE statement to refer to a file which is not in the open mode, but does not define what happens if the file is not in the open mode. The new I-O status value of 42 permits the user to test for this condition.

l. I-O status 43. For a mass storage file in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of a DELETE or REWRITE statement was not a successfully executed READ statement.

Second Standard COBOL specifies that for a file in sequential access mode, the last input-output statement executed for the file prior to the execution of a DELETE or REWRITE statement must have been a successfully executed READ statement, but it does not specify what happens if the requirement is not satisfied. The new I-O status value of 43 allows the user to test for this condition.

m. I-O status 44. A boundary violation exists because an attempt is made to rewrite a record to a sequential file and the record is not the same size as the record being replaced.

Second Standard COBOL specifies for a REWRITE statement that the number of character positions in the new record must be equal to the number of character positions in the record being replaced, but it does not specify what happens if this requirement is not satisfied. The new I-O status value of 44 allows the user to test for this condition.

n. I-O STATUS 46. A sequential READ statement is attempted on a file opened in the input or I-O mode and no valid next record has been established because either: (1) the preceding START statement was unsuccessful, (2) the preceding READ statement was unsuccessful but did not cause an at end condition, or (3) the preceding READ statement caused an at end condition.

Second Standard COBOL specifies that in these circumstances execution of a READ statement was illegal or its execution was unsuccessful, but did not specify a status code to indicate the situation. I-O status 46 can occur only if no corrective action is taken following the previous READ or START statement.

o. I-O status 47. The execution of a READ or START statement is attempted on a file not opened in the input or I-O mode.

Second Standard COBOL requires that the file must be opened in the input or I-O mode at the time a READ or START statement is executed, but does not specify what happens if the requirement is not met. The new I-O status value of 47 allows the user to test for this condition.

p. I-O status 48. The execution of a WRITE statement is attempted on either: (1) a sequential file not opened in the output or extend mode, or (2) a relative or indexed file not opened in the I-O, output, or extend mode.

Second Standard COBOL requires that the file be opened in one of the modes specified, but does not specify what happens if the requirement is not met. The new I-O status value of 48 allows the user to test for this condition.

q. I-O status 49. The execution of a DELETE or REWRITE statement is attempted on a file not opened in the I-O mode.

Second Standard COBOL requires that the file be opened in the I-O mode, but does not specify what happens if the requirement is not met. The new I-O status value of 49 allows the user to test for this condition.

41. Communication status key. New communication status key values have been added.

Second Standard COBOL leaves the results of some communication situations undefined. Third Standard COBOL defines new communication status key values for these situations so that the user can check for these error conditions in a standard way and thus take corrective action if appropriate.

These new communication status key values only affect existing programs which rely on some other action taking place when the newly defined exception condition occur.
The individual communication status key values added are described below.

a. Communication status key 15. Symbolic source, or one or more queues or destinations already disabled/enabled.

If, at the time a DISABLE or ENABLE statement is executed, the source or a queue or a destination referenced is already disabled or enabled respectively, the second Standard COBOL specifications imply that a communication status key value of 00 should be expected. The new communication status key value of 15 provides this information to the user.

b. Communication status key 21. Symbolic source is unknown.

In second Standard COBOL, the user has to compare the symbolic source data item with spaces to determine whether the symbolic name of the source terminal is known to the message control system (MCS) on a RECEIVE statement. Second Standard COBOL does not specify what happens if the symbolic source in an input CD referenced in an ENABLE or DISABLE statement is unknown. The new communication status key value of 21 provides this information.

c. Communication status key 65. Output queue capacity exceeded.

Second Standard COBOL does not specify what happens if the capacity of the output queue is exceeded on a SEND statement. This situation is now defined to give the new communication status key value of 65.

d. Communication status key 70. One or more destinations do not have portions associated with them.

This communication status key value is only returned by the new PURGE statement. Thus it cannot occur in programs written according to second Standard COBOL.

e. Communication status key 80. A combination of at least two status key conditions 10, 15, and 20 has occurred.

If the multiple destination facility is used and one of the destinations is disabled while a second destination is unknown, second Standard COBOL does not specify whether communication status key value 10 or 20 should be returned by a SEND statement. The new communication status key value of 80 is now defined to be returned in this situation. The new communication status key value of 80 is also returned in the case of an ENABLE or DISABLE statement where new communication status key condition 15 and communication status key 20 both apply.

42. Communication error key. New communication error key values have been added. These new communication error key values are described below.

a. Communication error key 2. Symbolic destination disabled.

A SEND statement was executed and the destination to which this error key applies is disabled. In second Standard COBOL, this condition was not distinguishable by the user.

b. Communication error key 5. Symbolic destination already enabled/disabled.

An ENABLE or DISABLE statement was executed and the destination to which this communication error key value applies was already enabled/disabled. In second Standard COBOL, this condition was not distinguishable by the user.

c. Communication error key value 6. Output queue capacity exceeded.

A SEND statement was executed and the MCS was not able to enqueue the message, message segment, or portion of the message or message segment because the output queue for the destination to which this communication error key value applies was full. In second Standard COBOL, this condition was not distinguishable by the user.

F. Composite Language Skeleton

F.1 GENERAL DESCRIPTION

This appendix contains the composite language skeleton of the DPS 7000 COBOL. It is intended to display complete and syntactically correct formats.

GENERAL FORMAT FOR CONTROL-DIVISION

```
|---------------------------------------------|
|  CONTROL DIVISION.                          |
| [SUBSTITUTION SECTION. [replace-entry]]     |
| [DEFAULT SECTION. [[default-entry]... . ]]  |
|---------------------------------------------|
```

GENERAL FORMAT FOR IDENTIFICATION-DIVISION

```
   IDENTIFICATION DIVISION.
                                {COMMON [INITIAL]}
   PROGRAM-ID. program-name [IS {               } PROGRAM].
                                {INITIAL [COMMON]}
  [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]]
  [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]... . ]]]
```

GENERAL FORMAT FOR DATA-DIVISION

```
   DATA DIVISION.

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

  [WORKING-STORAGE SECTION.

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

|--------------------------------------|
| [CONSTANT SECTION.                   |
|      [77-level-description-entry]    |
|      [                          ]... ] |
|      [record-description-entry  ]    |
|--------------------------------------|

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

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

  [REPORT SECTION.
       [report-description-entry
            {report-group-description-entry}... ]... ]
```

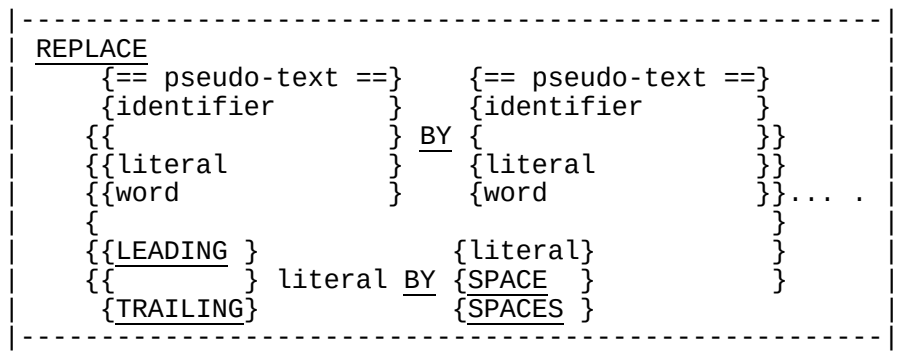
GENERAL FORMAT FOR PROCEDURE-DIVISION

```
   PROCEDURE DIVISION [USING {data-name}... ].

{ [DECLARATIVES.                                  }
{      {section-name SECTION [segment-number].    }
{       USE-statement.                            }
{      [paragraph-name.                           }
{      [sentence]... ]... }...                    }
{       END DECLARATIVES. ]                       }
{                                                 }
{      {section-name SECTION [segment-number].    }
{      [paragraph-name.                           }
{      [sentence]... ]... }...                    }
{                                                 }
{ {paragraph-name.                                }
{ [sentence]... }... ]                            }
```

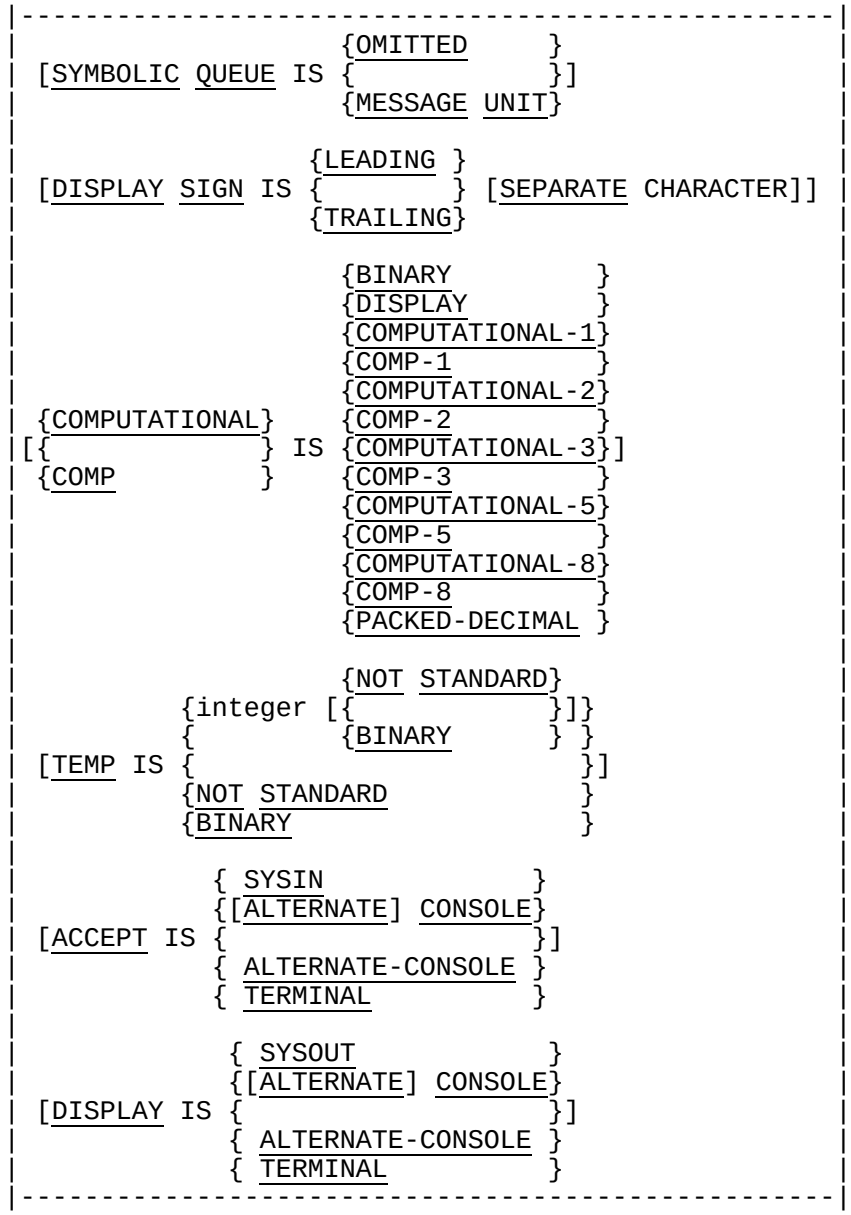
GENERAL FORMAT FOR END-PROGRAM-HEADER

```
  END PROGRAM program-name.
```

GENERAL FORMAT FOR REPLACE-ENTRY



GENERAL FORMAT FOR DEFAULT-ENTRIES

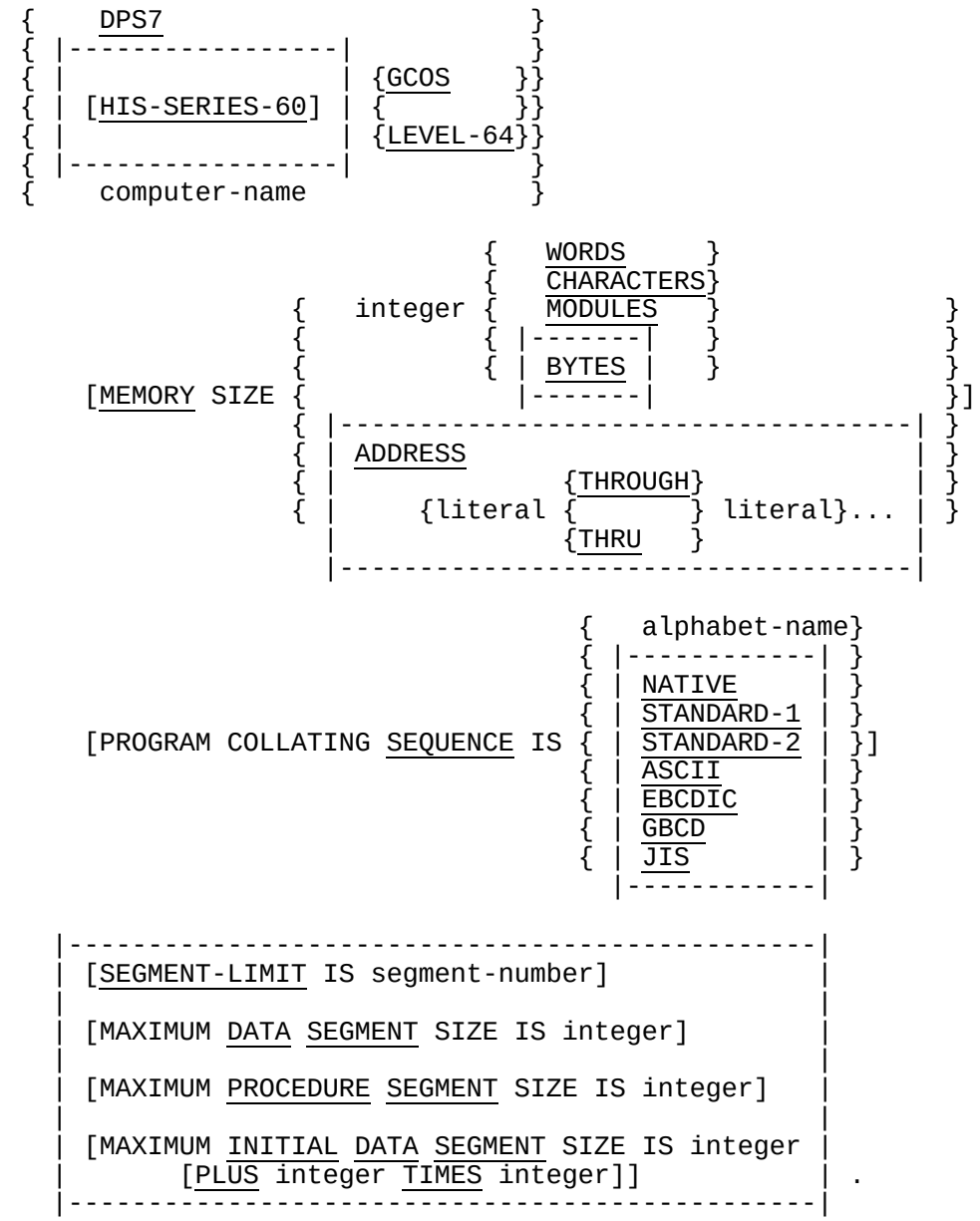


```
|-------------------------------------------------------|
| [SYSIN IS SYSIN-p]                                    |
|                                                       |
|         {ALTERNATE-CONSOLE}    {ALTERNATE-CONSOLE-p}  |
| [ACCEPT {                  } IS {                    }] |
|         {ALTERNATE CONSOLE}    {ALTERNATE CONSOLE-p}  |
|                                                       |
| [ACCEPT CONSOLE IS CONSOLE-p]                         |
|                                                       |
| [ACCEPT TERMINAL IS TERMINAL-p]                       |
|                                                       |
| [SYSOUT IS SYSOUT-p]                                  |
|                                                       |
|          {ALTERNATE-CONSOLE}    {ALTERNATE-CONSOLE-p} |
| [DISPLAY {                  } IS {                    }]|
|          {ALTERNATE CONSOLE}    {ALTERNATE CONSOLE-p} |
|                                                       |
| [DISPLAY CONSOLE IS CONSOLE-p]                        |
|                                                       |
| [DISPLAY TERMINAL IS TERMINAL-p]                      |
|                                                       |
|                  {FILE [COMMUNICATION]}               |
| [COBOL 1974 FOR {                     }]              |
|                  {COMMUNICATION [FILE]}               |
|-------------------------------------------------------|
```

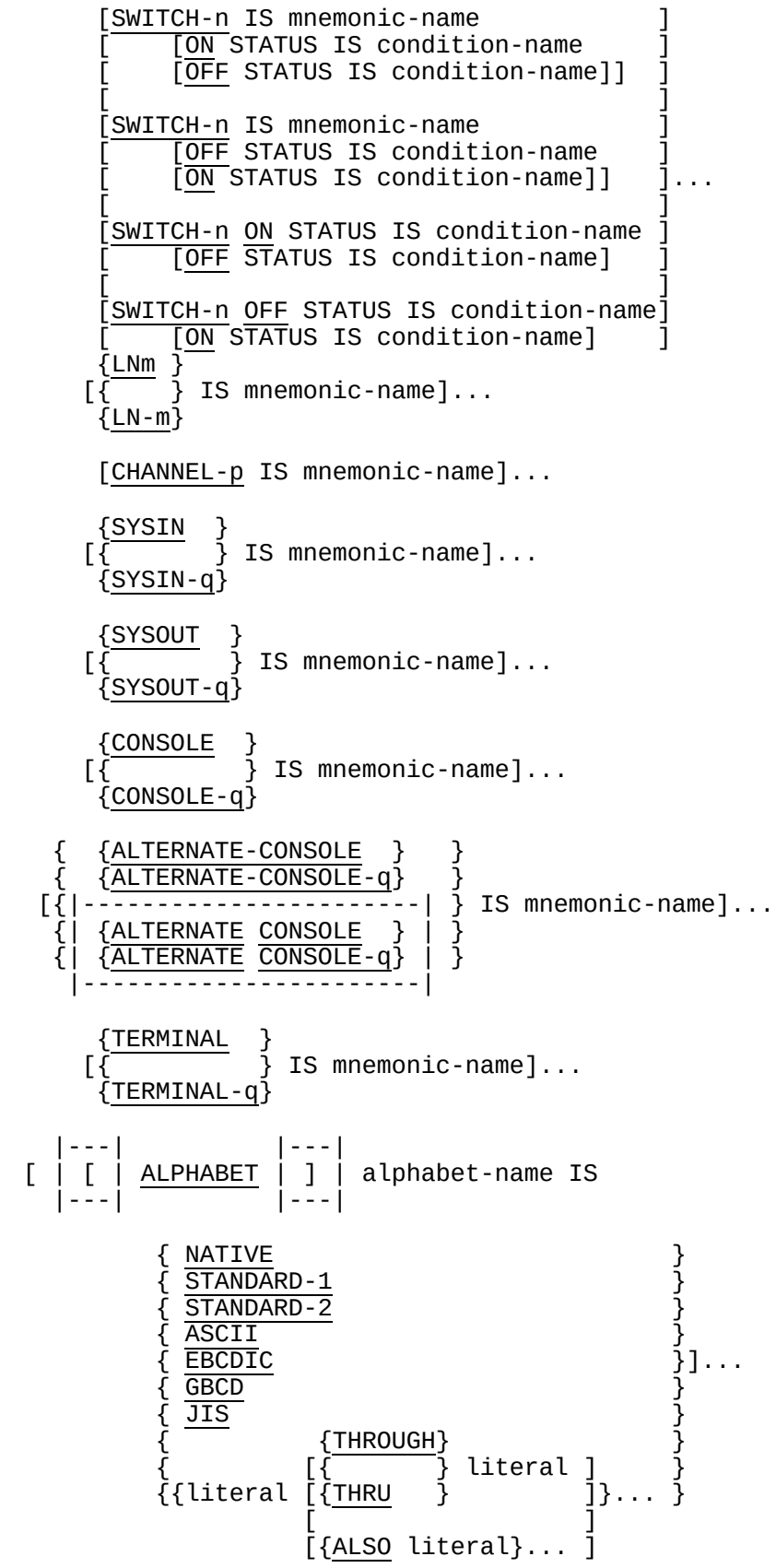
GENERAL FORMAT FOR SOURCE-COMPUTER-ENTRY

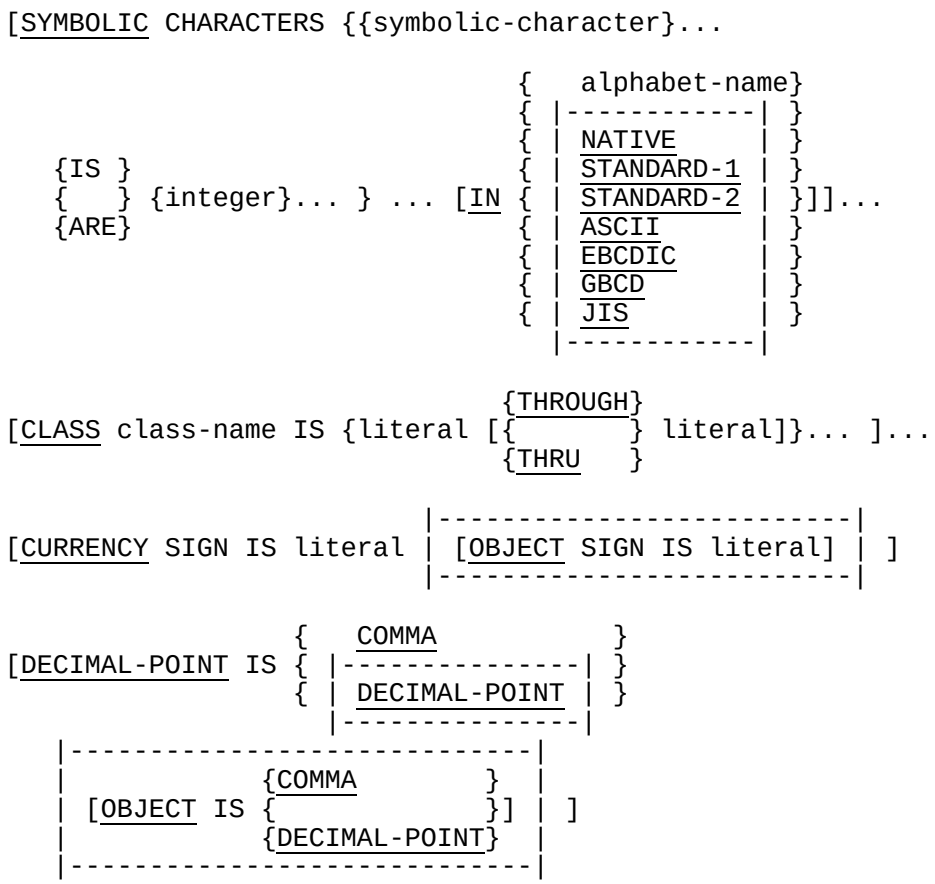
```
{    DPS7                      }
{ |-----------------|           }
{ |                 | {GCOS    }}
{ | [HIS-SERIES-60] | {        }}
{ |                 | {LEVEL-64}}
{ |-----------------|           }
{                               }
{    computer-name              }

   |-----------------------------------------------------|
   |                        {WORDS     }                 |
   |                        {CHARACTERS}                 |
   |              {integer {          }              }   |
   |              {        {MODULES   }              }   |
   |              {        {BYTES     }              }   |
   | [MEMORY SIZE {                                  }]  |
   |              {ADDRESS                           }   |
   |              {              {THROUGH}           }   |
   |              {    {literal {       } literal}... }   |
   |                             {THRU   }               |
   |-----------------------------------------------------|

     [WITH DEBUGGING MODE].
```

GENERAL FORMAT FOR OBJECT-COMPUTER-ENTRY

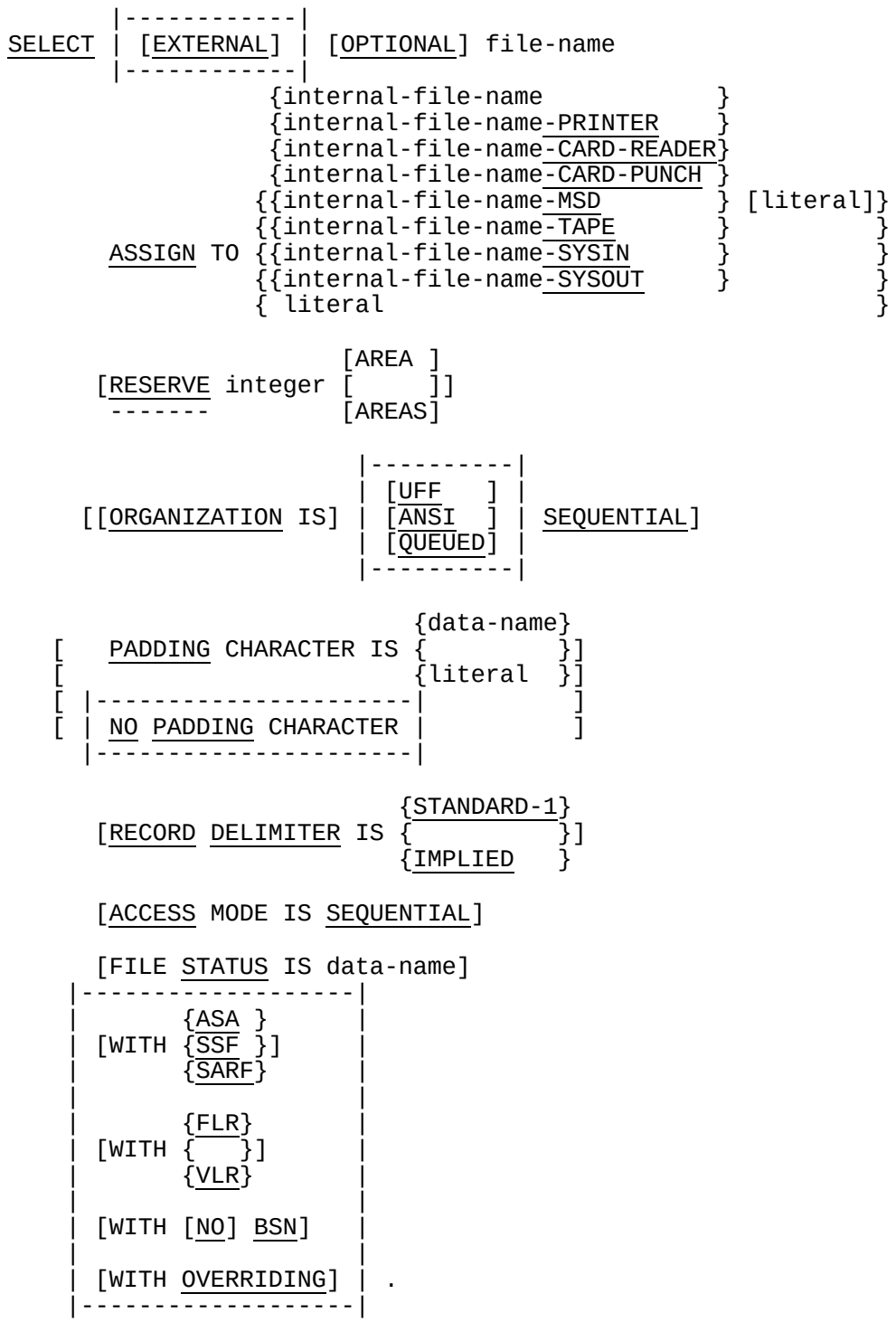


GENERAL FORMAT FOR SPECIAL-NAMES-ENTRIES

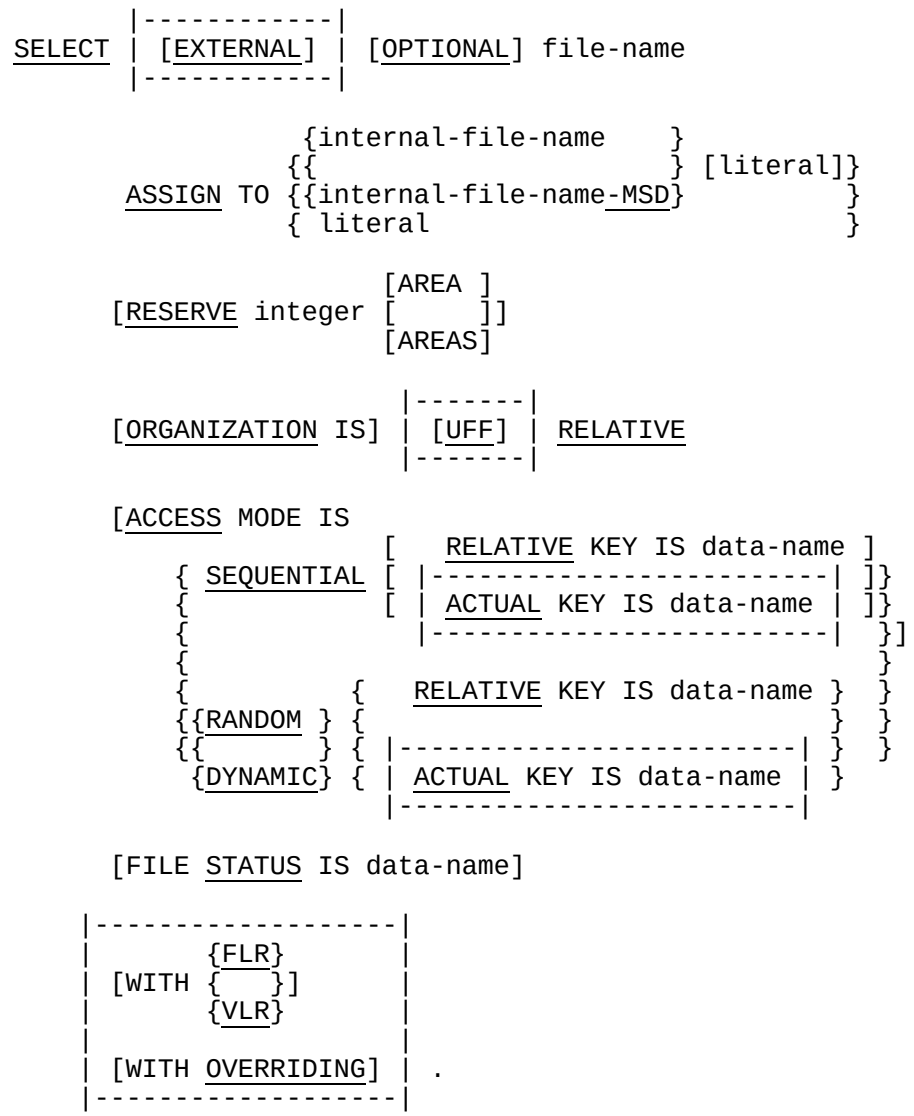



[SYMBOLIC CHARACTERS {{symbolic-character}...
{ alphabet-name}
{ NATIVE }
{IS } { STANDARD-1 }
{ } {integer}... } ... [IN { STANDARD-2 }]]...
{ARE} { ASCII }
{ EBCDIC }
{ GBCD }
{ JIS }
{THROUGH}
[CLASS class-name IS {literal [{ } literal]}...]...
{THRU }
[CURRENCY SIGN IS literal | [OBJECT SIGN IS literal] |]
[DECIMAL-POINT IS { COMMA }
{ DECIMAL-POINT }
[OBJECT IS {COMMA }] |]
{DECIMAL-POINT}

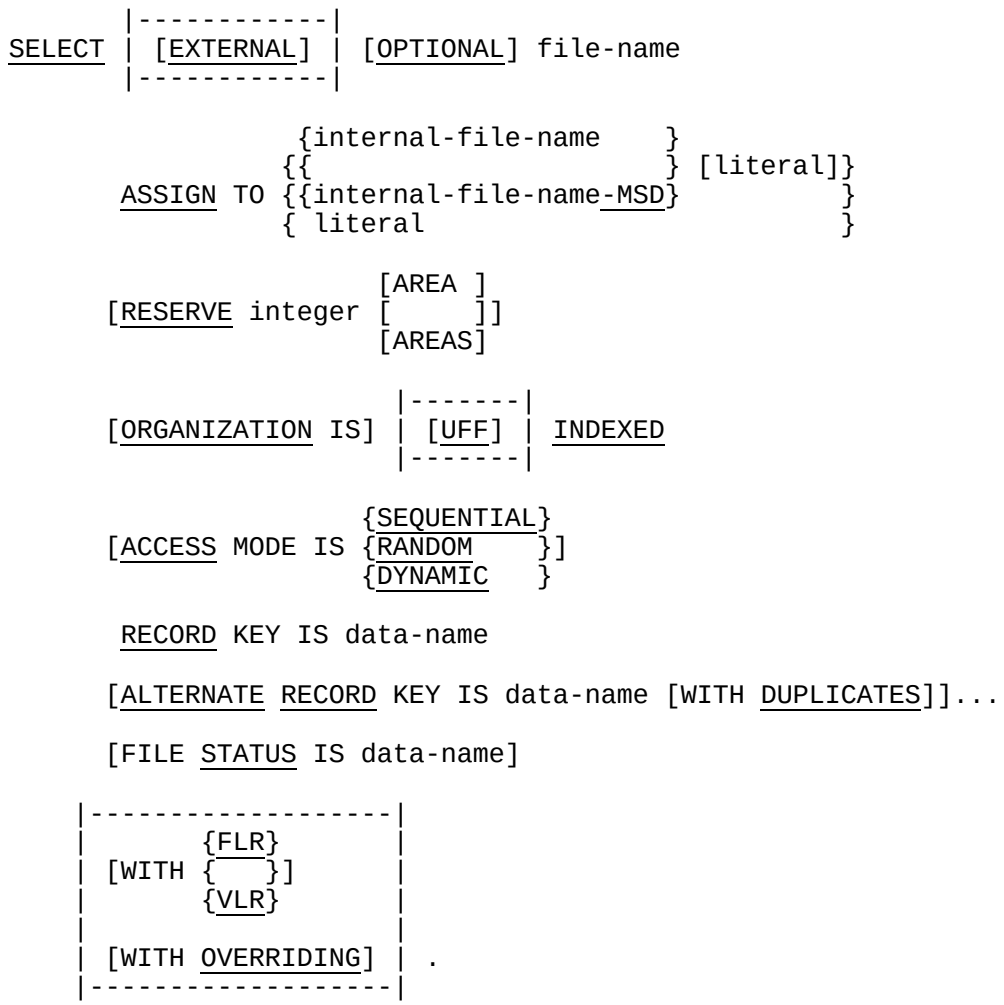
GENERAL FORMAT FOR FILE-CONTROL-ENTRY (SEQUENTIAL FILES)



GENERAL FORMAT FOR FILE-CONTROL-ENTRY (RELATIVE FILES)



GENERAL FORMAT FOR FILE-CONTROL-ENTRY (INDEXED FILES)



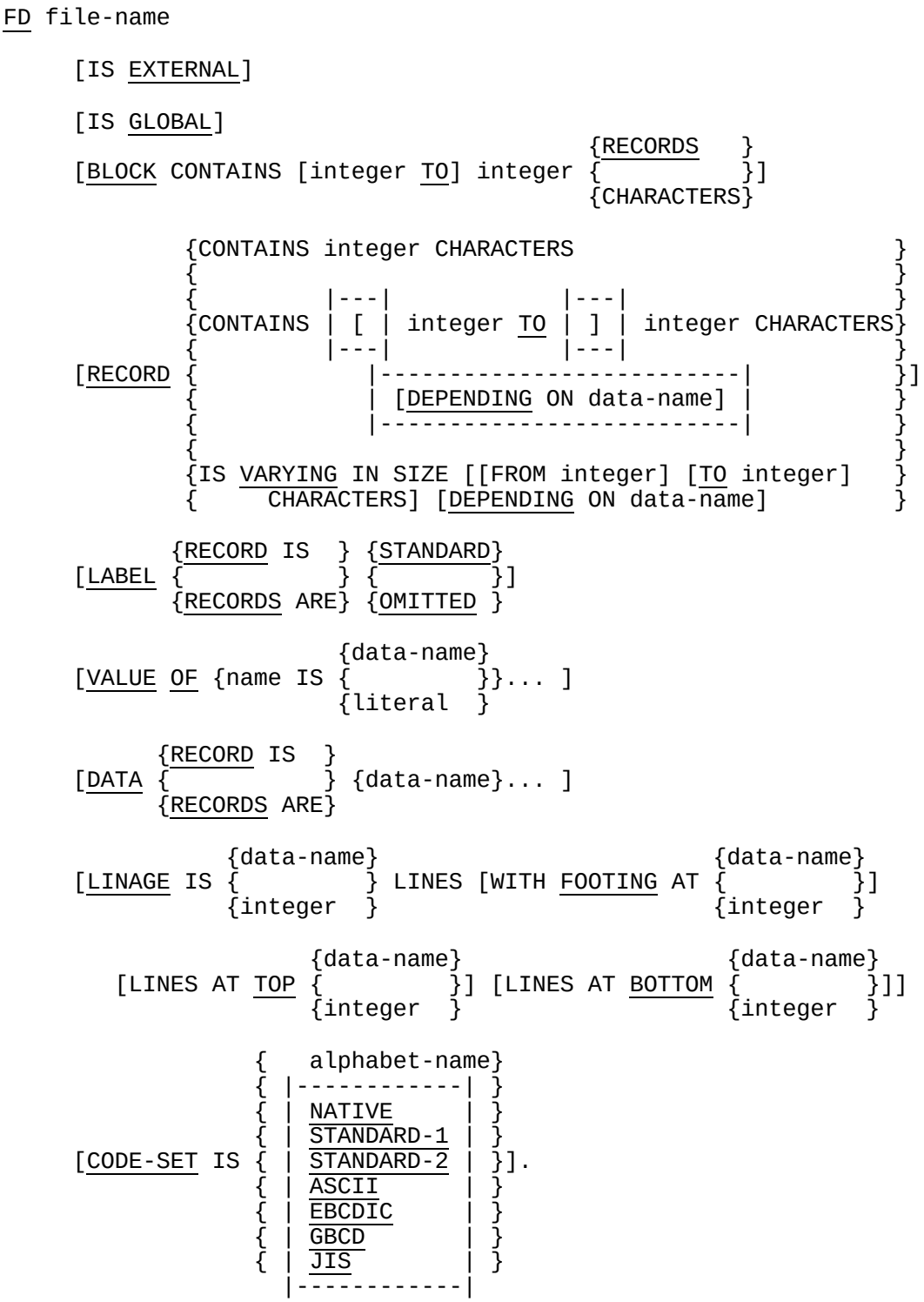
GENERAL FORMAT FOR FILE-CONTROL-ENTRY (SORT-MERGE FILES)

```
SELECT file-name

                {H-SORT                 }
       ASSIGN TO {internal-file-name    }
                 {internal-file-name-MSD}
                 {literal               }

    |--------------|
    |       {FLR}  |
    | [WITH {    }] | .
    |       {VLR}  |
    |--------------|
```

GENERAL FORMAT FOR INPUT-OUTPUT-CONTROL-ENTRIES

```
    |-----------------------------------------------|
    | [APPLY {NO-SORTED-INDEX ON {file-name}..}. ]. |
    |        {OPTIMIZE ON {file-name}         }     |
    |-----------------------------------------------|

      [RERUN ON CHECKPOINT-FILE
                   { integer RECORDS}
            EVERY {                } OF file-name]...
                  {         {REEL} }
                  {[END OF] {    } }
                            {UNIT}

            [RECORD    ]
      [SAME [SORT      ] AREA FOR file-name {file-name}... ]...
            [SORT-MERGE]

      [MULTIPLE FILE TAPE CONTAINS
           {file-name [POSITION integer]}... ]...
```

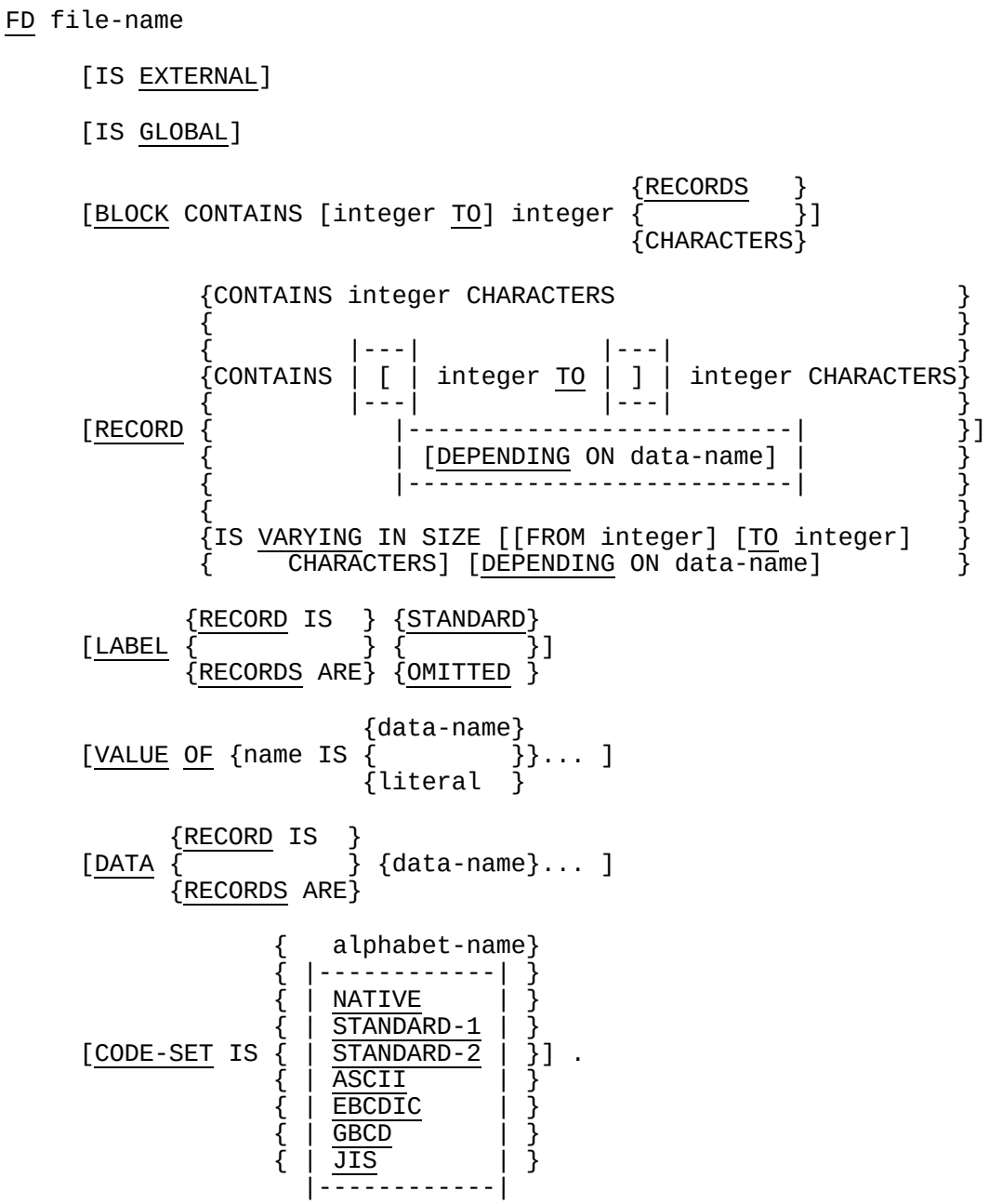
GENERAL FORMAT FOR SEQUENTIAL-FILE-DESCRIPTION-ENTRY



GENERAL FORMAT FOR REPORT-FILE-DESCRIPTION-ENTRY

```
FD file-name

      [IS EXTERNAL]

      [IS GLOBAL]
                                                 {RECORDS   }
      [BLOCK CONTAINS [integer TO] integer       {          }]
                                                 {CHARACTERS}

              {CONTAINS integer CHARACTERS                  }
      [RECORD {                                             }]
              {CONTAINS integer TO integer CHARACTERS       }

             {RECORD IS  } {STANDARD}
      [LABEL {           } {        }]
             {RECORDS ARE} {OMITTED }

                           {data-name}
      [VALUE OF {name IS   {         }}... ]
                           {literal   }

                    {   alphabet-name}
                    { |------------| }
                    { | NATIVE     | }
                    { | STANDARD-1 | }
      [CODE-SET IS  { | STANDARD-2 | }]
                    { | ASCII      | }
                    { | EBCDIC     | }
                    { | GBCD       | }
                    { | JIS        | }
                      |------------|
      {REPORT IS  }
      {           } {report-name}... .
      {REPORTS ARE}
```

GENERAL FORMAT FOR RELATIVE-FILE-DESCRIPTION-ENTRY



GENERAL FORMAT FOR INDEXED-FILE-DESCRIPTION-ENTRY

```
FD file-name

     [IS EXTERNAL]

     [IS GLOBAL]

                                                {RECORDS   }
     [BLOCK CONTAINS [integer TO] integer       {          }]
                                                {CHARACTERS}

             {CONTAINS integer CHARACTERS                          }
             {                                                     }
             {          |---|            |---|                    }
             {CONTAINS  | [ | integer TO | ] | integer CHARACTERS }
             {          |---|            |---|                    }
     [RECORD {            |--------------------------|            }]
             {            | [DEPENDING ON data-name] |            }
             {            |--------------------------|            }
             {                                                     }
             {IS VARYING IN SIZE [[FROM integer] [TO integer]      }
             {     CHARACTERS] [DEPENDING ON data-name]            }

            {RECORD IS  } {STANDARD}
     [LABEL {           } {         }]
            {RECORDS ARE} {OMITTED  }

                          {data-name}
     [VALUE OF {name IS  {         }}... ]
                          {literal  }

           {RECORD IS  }
     [DATA {           } {data-name}... ]
           {RECORDS ARE}

                          {   alphabet-name}
                          { |------------| }
                          { | NATIVE     | }
                          { | STANDARD-1 | }
           [CODE-SET IS   { | STANDARD-2 | }] .
                          { | ASCII      | }
                          { | EBCDIC     | }
                          { | GBCD       | }
                          { | JIS        | }
                            |------------|
```

GENERAL FORMAT FOR SORT-MERGE-FILE-DESCRIPTION-ENTRY

```
              |-------------|
SD file-name  | [IS GLOBAL] |
              |-------------|

              {CONTAINS integer CHARACTERS                      }
              {                                                 }
              {          |---|            |---|                 }
              {CONTAINS  | [ | integer TO | ] | integer CHARACTERS}
              {          |---|            |---|                 }
     [RECORD  {             |-------------------------|          }]
              {             | [DEPENDING ON data-name] |         }
              {             |-------------------------|          }
              {                                                 }
              {IS VARYING IN SIZE [[FROM integer] [TO integer]  }
              {     CHARACTERS] [DEPENDING ON data-name]        }

         {RECORD IS  }
     [DATA {         } {data-name}... ].
         {RECORDS ARE}
```

GENERAL FORMATS FOR COMMUNICATION-DESCRIPTION-ENTRY

Format 1

```
CD cd-name FOR [INITIAL] INPUT

    |-------------|
    | [IS GLOBAL] |
    |-------------|

    [[[SYMBOLIC QUEUE IS data-name]                    ]
    [                                                  ]
    [      [SYMBOLIC SUB-QUEUE-1 IS data-name]         ]
    [                                                  ]
    [      [SYMBOLIC SUB-QUEUE-2 IS data-name]         ]
    [                                                  ]
    [      [SYMBOLIC SUB-QUEUE-3 IS data-name]         ]
    [                                                  ]
    [      [MESSAGE DATE IS data-name]                 ]
    [                                                  ]
    [      [MESSAGE TIME IS data-name]                 ]
    [                                                  ]
    [      [SYMBOLIC SOURCE IS data-name]              ]
    [                                                  ]
    [      [TEXT LENGTH IS data-name]                  ].
    [                                                  ]
    [      [END KEY IS data-name]                      ]
    [                                                  ]
    [      [STATUS KEY IS data-name]                   ]
    [                                                  ]
    [      [MESSAGE COUNT IS data-name]]               ]
    [                                                  ]
    [ [data-name data-name data-name                   ]
    [       data-name data-name data-name              ]
    [       data-name data-name data-name              ]
    [       data-name data-name]                       ]
```

Format 2

```
CD cd-name FOR OUTPUT

    |-------------|
    | [IS GLOBAL] |
    |-------------|

      [DESTINATION COUNT IS data-name]

      [TEXT LENGTH IS data-name]

      [STATUS KEY IS data-name]

      [DESTINATION TABLE OCCURS integer TIMES
           [INDEXED BY {index-name}... ]]

      [ERROR KEY IS data-name]

      [SYMBOLIC DESTINATION IS data-name].
```

Format 3

```
CD cd-name FOR [INITIAL] I-O

    |-------------|
    | [IS GLOBAL] |
    |-------------|

    [[[MESSAGE DATE IS data-name]          ]
    [                                      ]
    [      [MESSAGE TIME IS data-name]     ]
    [                                      ]
    [      [SYMBOLIC TERMINAL IS data-name]]
    [                                      ]
    [      [TEXT LENGTH IS data-name]      ] .
    [                                      ]
    [      [END KEY IS data-name]          ]
    [                                      ]
    [      [STATUS KEY IS data-name]]      ]
    [                                      ]
    [ [data-name data-name data-name       ]
    [       data-name data-name data-name] ]
```

GENERAL FORMAT FOR REPORT-DESCRIPTION-ENTRY

```
RD report-name [IS GLOBAL]

      [CODE literal]

      {CONTROL IS  } {{data-name}...        }
     [{            } {                      }]
      {CONTROLS ARE} { FINAL [data-name]... }

            [LIMIT IS  ]         [LINE ]
      [PAGE [          ] integer [     ] [HEADING integer]
            [LIMITS ARE]         [LINES]

           [FIRST DETAIL integer] [LAST DETAIL integer]

           [FOOTING integer]].
```

GENERAL FORMATS FOR DATA-DESCRIPTION-ENTRY

Format 1

```
             [data-name]
level-number [         ]
             [FILLER   ]

    [REDEFINES data-name]

    [IS EXTERNAL]

    [IS GLOBAL]

    {PICTURE}                        |--------------------------|
   [{       } IS character-string | [DEPENDING ON data-name] |]
    {PIC    }                        |--------------------------|

                 {   BINARY             }
                 { |-----|              }
                 { | BIT |              }
                 { |-----|              }
                 {   COMPUTATIONAL      }
                 {   COMP               }
                 { |------------------| }
                 { | COMPUTATIONAL-1  | }
                 { | COMP-1           | }
                 { | COMPUTATIONAL-2  | }
                 { | COMP-2           | }
                 { | COMPUTATIONAL-3  | }
                 { | COMP-3           | }
   [[USAGE IS]   { | COMPUTATIONAL-5  | }]
                 { | COMP-5           | }
                 { | COMPUTATIONAL-8  | }
                 { | COMP-8           | }
                 { | COMPUTATIONAL-9  | }
                 { | COMP-9           | }
                 { | COMPUTATIONAL-10 | }
                 { | COMP-10          | }
                 { | COMPUTATIONAL-15 | }
                 { | COMP-15          | }
                 { | POINTER          | }
                 { |------------------| }
                 {   DISPLAY            }
                 {   INDEX              }
                 {   PACKED-DECIMAL     }

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

```
    [OCCURS integer TIMES                                          ]
    [                                                              ]
    [    {ASCENDING }                                              ]
    [   [{          } KEY IS {data-name}... ]...                   ]
    [    {DESCENDING}                                              ]
    [                                                              ]
    [    [INDEXED BY {index-name}... ]                             ]
    [                                                              ]
    [OCCURS integer TO integer TIMES DEPENDING ON data-name]
    [                                                              ]
    [    {ASCENDING }                                              ]
    [   [{          } KEY IS {data-name}... ]...                   ]
    [    {DESCENDING}                                              ]
    [                                                              ]
    [    [INDEXED BY {index-name}... ]                             ]

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

    {JUSTIFIED}
   [{         } RIGHT]
    {JUST     }

    [BLANK WHEN ZERO]

              {   literal}
    [VALUE IS { |------| }] .
              { | NULL | }
                |------|
```

Format 2

```
66 data-name
                         {THROUGH}
     RENAMES data-name [{        } data-name].
                         {THRU   }
```

Format 3

```
88 condition-name
                   |------|
                 { | NULL |                                  }
    {VALUE IS  } { |------|                                  }
    {          } {            {THROUGH}           |-----| }
    {VALUES ARE} {  {literal [{-------} literal]} | ... | }
                 {            {THRU   }           |-----| }

      |---------------------------------|
      | [WHEN SET TO FALSE IS literal] | .
      |---------------------------------|
```

GENERAL FORMATS FOR REPORT-GROUP-DESCRIPTION-ENTRY

Format 1

```
01 [data-name]
                    {integer [ON NEXT PAGE]}
    [LINE NUMBER IS {                       }]
                    {PLUS integer           }

                   {integer     }
    [NEXT GROUP IS {PLUS integer}]
                   {NEXT PAGE   }

              {{REPORT HEADING}             }
              {{RH            }             }
              {                             }
              {{PAGE HEADING}               }
              {{PH           }              }
              {                             }
              {{CONTROL HEADING} {data-name}}
              {{CH              } {FINAL    }}
              {                             }
     TYPE IS {{DETAIL}                      }
              {{DE    }                     }
              {                             }
              {{CONTROL FOOTING} {data-name}}
              {{CF              } {FINAL    }}
              {                             }
              {{PAGE FOOTING}               }
              {{PF           }              }
              {                             }
              {{REPORT FOOTING}             }
              {{RF            }             }

   [[USAGE IS] DISPLAY].
```

Format 2

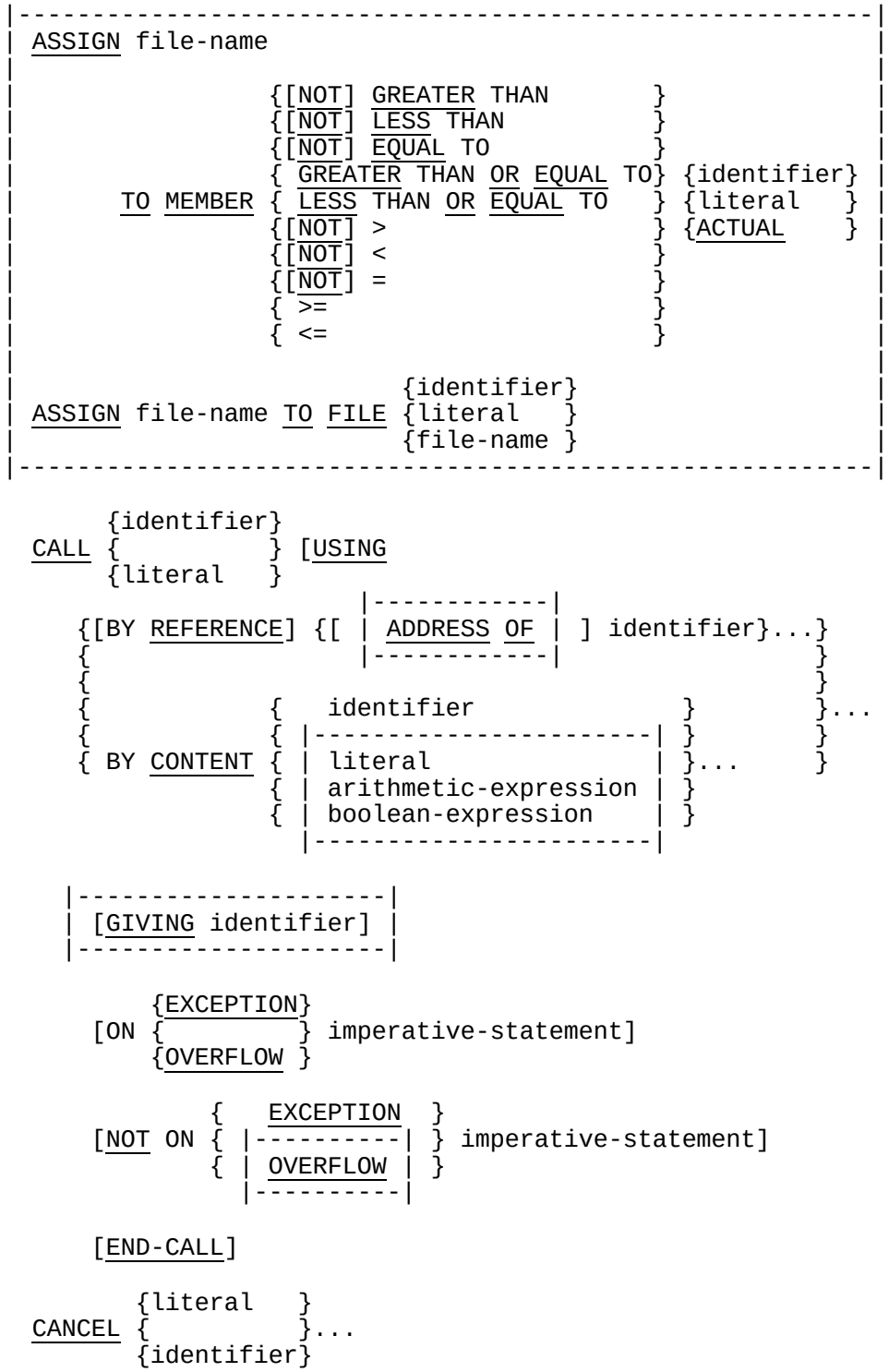
```
level-number [data-name]

                    {integer [ON NEXT PAGE]}
    [LINE NUMBER IS {                       }]
                    {PLUS integer           }

   [[USAGE IS] DISPLAY].
```

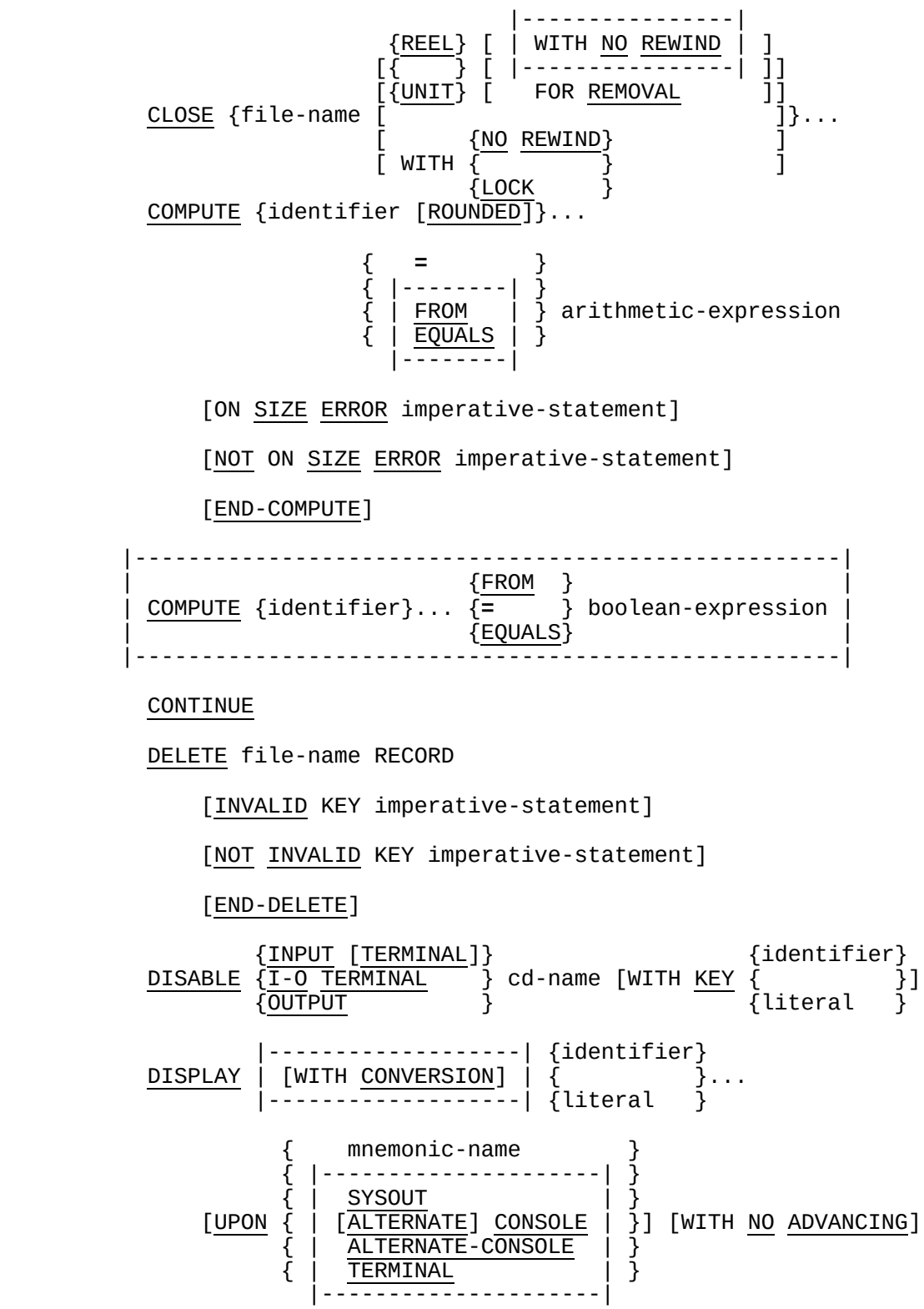
Format 3

```
level-number [data-name]

    {PICTURE}
    {       } IS character-string
    {PIC    }

   [[USAGE IS] DISPLAY]

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

    {JUSTIFIED}
   [{         } RIGHT]
    {JUST     }

    [BLANK WHEN ZERO]

                    {integer [ON NEXT PAGE]}
    [LINE NUMBER IS {                      }]
                    {PLUS integer          }

    [COLUMN NUMBER IS integer]

   { SOURCE IS identifier                          }
   {                                               }
   { VALUE IS literal                              }
   {                                               }
   {{SUM {identifier}... [UPON {data-name}... ]}...}
   {                                               }
   {                {data-name}                    }
   {     [RESET ON {         }]                    }
                    {FINAL    }

    [GROUP INDICATE].
```

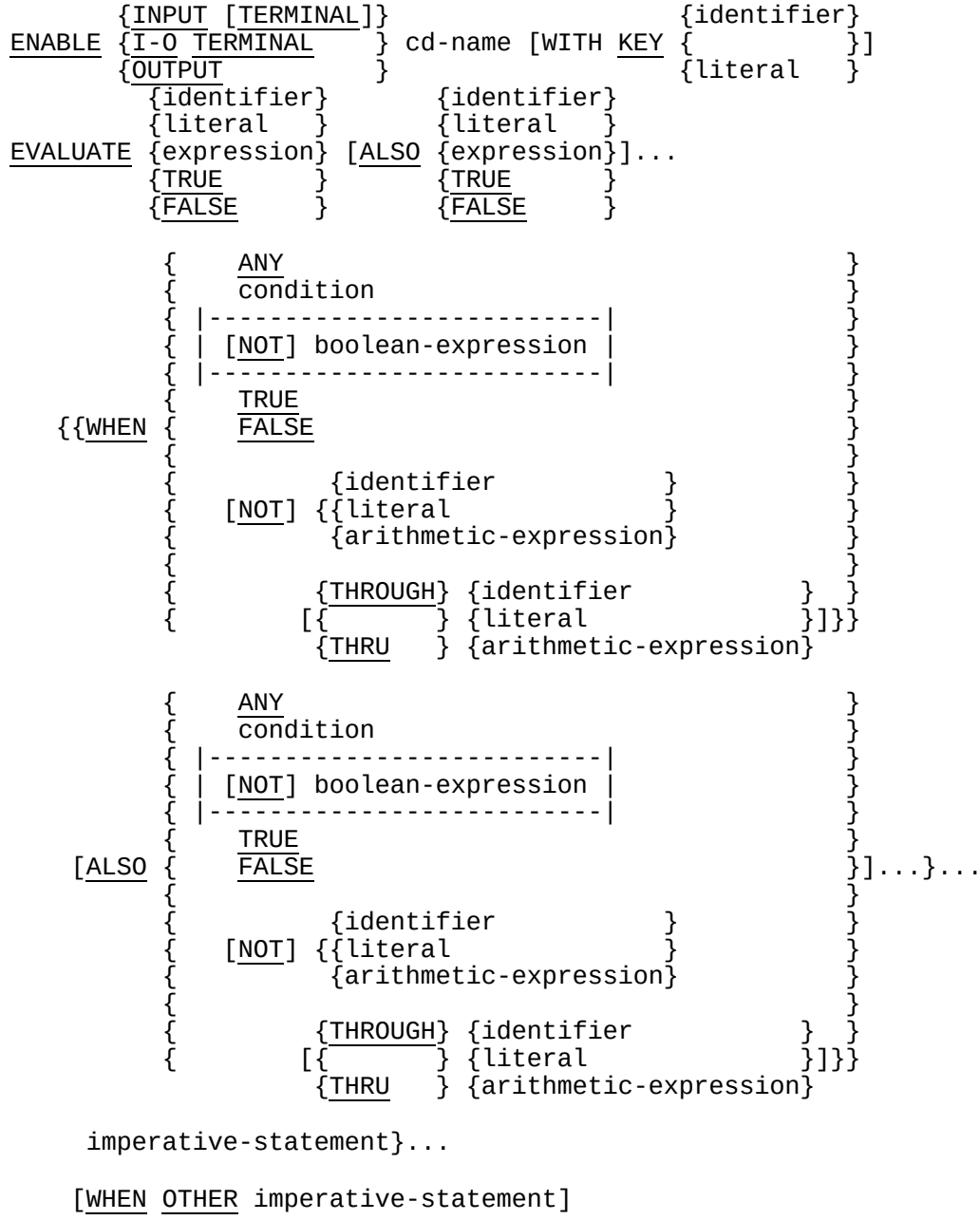
GENERAL FORMAT FOR VERBS

```
                            {    mnemonic-name        }
                            { |---------------------| }
                            { |  SYSIN               | }
  ACCEPT identifier [FROM   { | [ALTERNATE] CONSOLE  | }
                            { |  ALTERNATE-CONSOLE   | }
                            { |  TERMINAL            | }
                              |---------------------|

                           {DATE       }
                           {DAY        }
  ACCEPT identifier FROM   {           }
                           {DAY-OF-WEEK}
                           {TIME       }
  ACCEPT cd-name MESSAGE COUNT

|----------------------------------------|
| ACCEPT identifier FROM FILE file-name  |
|----------------------------------------|

      {identifier}
  ADD {          }... TO {identifier [ROUNDED]}...
      {literal   }

      [ON SIZE ERROR imperative-statement]

      [NOT ON SIZE ERROR imperative-statement]

      [END-ADD]

     {identifier}     {identifier}
  ADD{          }...TO{          } GIVING {identifier[ROUNDED]}...
     {literal   }     {literal   }

      [ON SIZE ERROR imperative-statement]

      [NOT ON SIZE ERROR imperative-statement]

      [END-ADD]

      {CORRESPONDING}
  ADD {              } identifier TO identifier [ROUNDED]
      {CORR          }

      [ON SIZE ERROR imperative-statement]

      [NOT ON SIZE ERROR imperative-statement]

      [END-ADD]

  ALTER {procedure-name TO [PROCEED TO] procedure-name}...
```

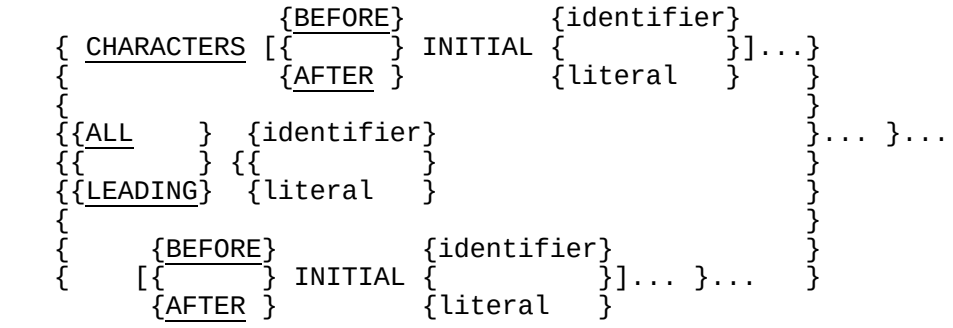

ASSIGN file-name
{[NOT] GREATER THAN }
{[NOT] LESS THAN }
{[NOT] EQUAL TO }
{ GREATER THAN OR EQUAL TO} {identifier}
TO MEMBER { LESS THAN OR EQUAL TO } {literal }
{[NOT] > } {ACTUAL }
{[NOT] < }
{[NOT] = }
{ >= }
{ <= }
{identifier}
ASSIGN file-name TO FILE {literal }
{file-name }
{identifier}
CALL {literal } [USING
{[BY REFERENCE] {[| ADDRESS OF |] identifier}...}
{ BY CONTENT { identifier } }...]
{ literal }...
{ arithmetic-expression }
{ boolean-expression }
[GIVING identifier]
{EXCEPTION}
[ON {OVERFLOW } imperative-statement]
{ EXCEPTION }
[NOT ON { OVERFLOW } imperative-statement]
[END-CALL]
{literal }
CANCEL {identifier}...


CLOSE {file-name [{REEL} [| WITH NO REWIND |] {UNIT} [FOR REMOVAL]]}...
[{NO REWIND}]
[WITH { }]
{LOCK }
COMPUTE {identifier [ROUNDED]}...
{ = }
{ | FROM | } arithmetic-expression
{ | EQUALS | }
[ON SIZE ERROR imperative-statement]
[NOT ON SIZE ERROR imperative-statement]
[END-COMPUTE]
{FROM }
COMPUTE {identifier}... {= } boolean-expression
{EQUALS}
CONTINUE
DELETE file-name RECORD
[INVALID KEY imperative-statement]
[NOT INVALID KEY imperative-statement]
[END-DELETE]
{INPUT [TERMINAL]} {identifier}
DISABLE {I-O TERMINAL } cd-name [WITH KEY { }]
{OUTPUT } {literal }
| [WITH CONVERSION] | {identifier}...
DISPLAY {literal }
{ mnemonic-name }
{ SYSOUT }
[UPON { [ALTERNATE] CONSOLE }] [WITH NO ADVANCING]
{ ALTERNATE-CONSOLE }
{ TERMINAL }

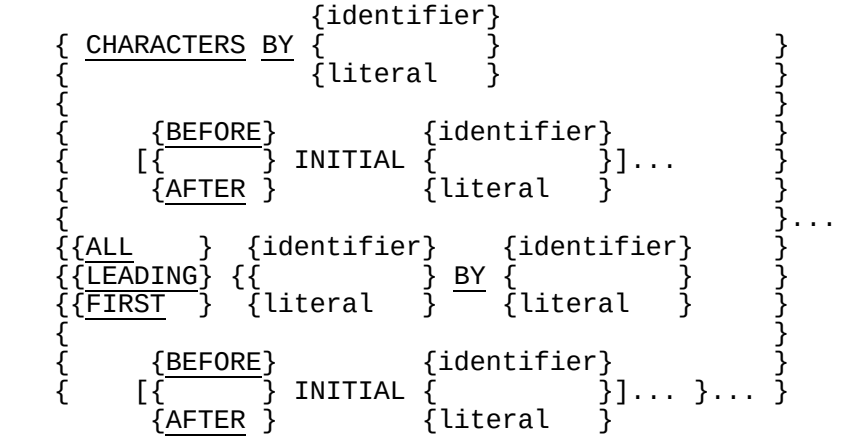
```
        {identifier}
DIVIDE  {           } INTO {identifier [ROUNDED]}...
        {literal   }

    [ON SIZE ERROR imperative-statement]

    [NOT ON SIZE ERROR imperative-statement]

    [END-DIVIDE]

        {identifier}      {identifier}
DIVIDE  {           } INTO {          }
        {literal   }      {literal   }

     GIVING {identifier [ROUNDED]}...

    [ON SIZE ERROR imperative-statement]

    [NOT ON SIZE ERROR imperative-statement]

    [END-DIVIDE]

        {identifier}    {identifier}
DIVIDE  {           } BY {          }
        {literal   }    {literal   }

                        GIVING {identifier [ROUNDED]}...

    [ON SIZE ERROR imperative-statement]

    [NOT ON SIZE ERROR imperative-statement]

    [END-DIVIDE]

        {identifier}      {identifier}
DIVIDE  {           } INTO {          }
        {literal   }      {literal   }

     GIVING identifier [ROUNDED] REMAINDER identifier

    [ON SIZE ERROR imperative-statement]

    [NOT ON SIZE ERROR imperative-statement]

    [END-DIVIDE]

        {identifier}    {identifier}
DIVIDE  {           } BY {          }
        {literal   }    {literal   }

     GIVING identifier [ROUNDED] REMAINDER identifier

    [ON SIZE ERROR imperative-statement]

    [NOT ON SIZE ERROR imperative-statement]

    [END-DIVIDE]
```


{INPUT [TERMINAL]}
ENABLE {I-O TERMINAL }
{OUTPUT }
{identifier}
{literal }
cd-name [WITH KEY {identifier}
{literal }]
EVALUATE {identifier} {identifier}
{literal } {literal }
{expression} [ALSO {expression}]...
{TRUE } {TRUE }
{FALSE } {FALSE }
{{WHEN {ANY
condition
[NOT] boolean-expression
TRUE
FALSE
{identifier }
[NOT] {{literal }
{arithmetic-expression}
{THROUGH} {identifier }
[{ } {literal }]}}
{THRU } {arithmetic-expression}
[ALSO {ANY
condition
[NOT] boolean-expression
TRUE
FALSE
{identifier }
[NOT] {{literal }
{arithmetic-expression}
{THROUGH} {identifier }
[{ } {literal }]}}...}...
{THRU } {arithmetic-expression}
imperative-statement}...
[WHEN OTHER imperative-statement]

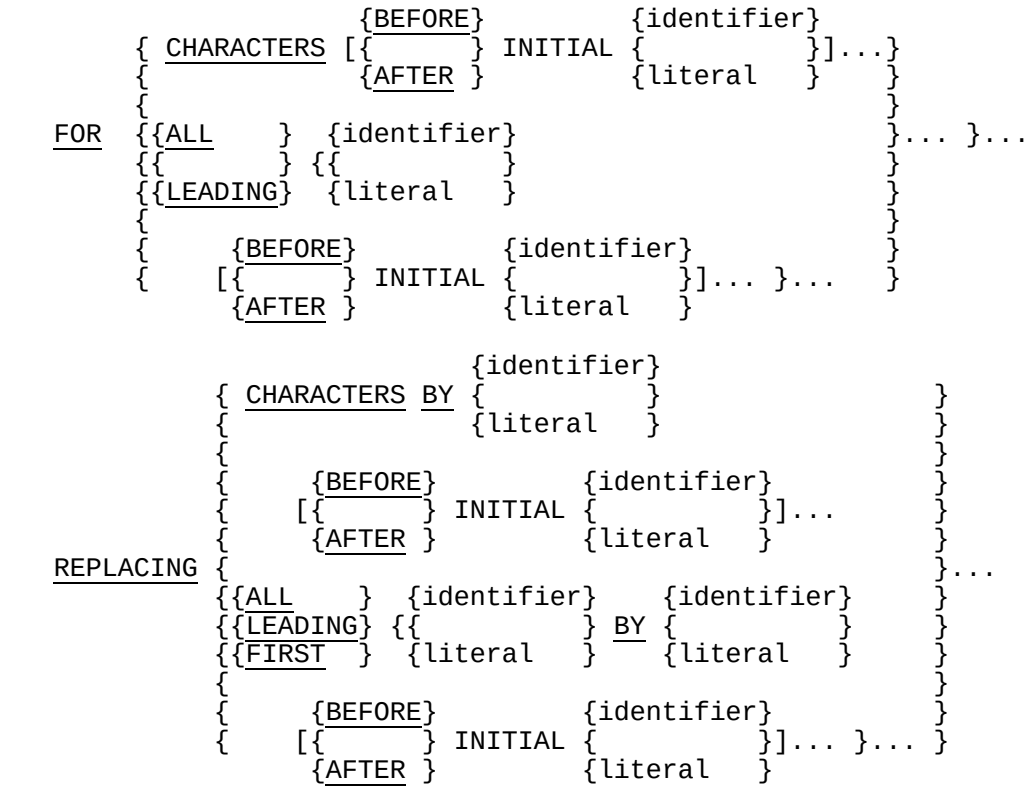
```
     [END-EVALUATE]
|-------------------------------------------------------------|
|                                                             |
| EXAMINE identifier                                          |
|                                                             |
|               {ALL        } {literal   }                    |
|     {TALLYING {LEADING    } {          }                  } |
|     {         {UNTIL FIRST} {identifier}                  } |
|     {                                                     } |
|     {                   {literal   }                      } |
|     {    [REPLACING BY {          }]                      } |
|     {                   {identifier}                      } |
|     {                                                     } |
|     {          { ALL         } {literal   }    {literal   }} |
|     {REPLACING { LEADING     } {          } BY {          }} |
|     {          {[UNTIL] FIRST} {identifier}    {identifier}} |
|                                                             |
|-------------------------------------------------------------|
 EXIT [PROGRAM [GIVING identifier]].

          {data-name  }
 GENERATE {           }
          {report-name}

 GO TO [procedure-name]

 GO TO {procedure-name}... DEPENDING ON identifier

                   {{statement}...} {ELSE {statement}...[END-IF]}
 IF condition THEN {              } {ELSE NEXT SENTENCE         }
                   { NEXT SENTENCE} {END-IF                     }

 INITIALIZE {identifier}...

              {   ALPHABETIC         }
              { |---------|          }
              { | BOOLEAN |          }          {identifier}
   [REPLACING { |---------|          } DATA BY {          }... ]
              {   ALPHANUMERIC       }          {literal   }
              {   NUMERIC            }
              {   ALPHANUMERIC-EDITED}
              {   NUMERIC-EDITED     }

 INITIATE {report-name}...

 INSPECT identifier TALLYING {identifier FOR
```



INSPECT identifier REPLACING



INSPECT identifier TALLYING {identifier

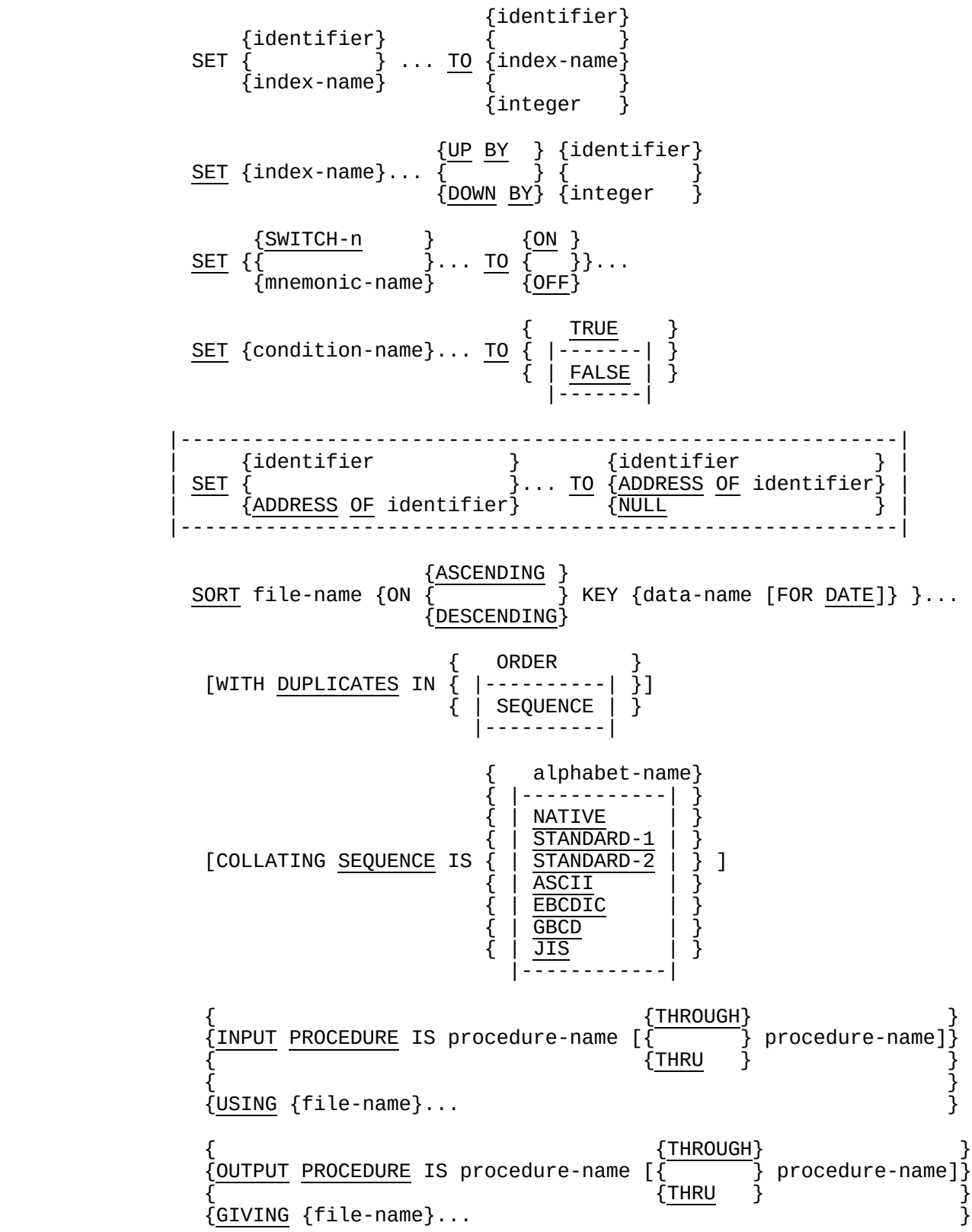


```
                                 {identifier}    {identifier}
INSPECT identifier CONVERTING    {          } TO {          }
                                 {literal   }    {literal   }

         {BEFORE}         {identifier}
        [{      } INITIAL {          }]...
         {AFTER }         {literal   }

                      {ASCENDING }
MERGE file-name {ON   {          } KEY {data-name [FOR DATE]} }...
                      {DESCENDING}

                           {   alphabet-name}
                           { |------------| }
                           { | NATIVE     | }
                           { | STANDARD-1 | }
                           { | STANDARD-2 | }]
  [COLLATING SEQUENCE IS   { | ASCII      | }]
                           { | EBCDIC     | }
                           { | GBCD       | }
                           { | JIS        | }
                             |------------|

   USING file-name {file-name}...

  {                                        {THROUGH}                }
  {OUTPUT PROCEDURE IS procedure-name[{       } procedure-name]}
  {                                        {THRU   }                }
  {                                                                 }
  {GIVING {file-name}...                                            }

     {identifier}
MOVE {          } TO {identifier}...
     {literal   }

     {CORRESPONDING}
MOVE {             } identifier TO {identifier} ...
     {CORR         }

         {identifier}
MULTIPLY {          } BY {identifier [ROUNDED]}...
         {literal   }

    [ON SIZE ERROR imperative-statement]

    [NOT ON SIZE ERROR imperative-statement]

    [END-MULTIPLY]
```

```
          {identifier}    {identifier}
MULTIPLY  {          } BY {          }
          {literal   }    {literal   }

     GIVING {identifier [ROUNDED]}...

    [ON SIZE ERROR imperative-statement]

    [NOT ON SIZE ERROR imperative-statement]

    [END-MULTIPLY]

     {INPUT {file-name [WITH NO REWIND]}... }
OPEN {OUTPUT {file-name [WITH NO REWIND]}...} ...
     {I-O {file-name}...                      }
     {EXTEND {file-name}...                   }

                             {THROUGH}
PERFORM [procedure-name [{       } procedure-name]]
                             {THRU   }

    [imperative-statement END-PERFORM]

                             {THROUGH}
PERFORM [procedure-name [{       } procedure-name]]
                             {THRU   }

    {identifier}
    {          } TIMES
    {integer   }

    [imperative-statement END-PERFORM]

                             {THROUGH}
PERFORM [procedure-name [{       } procedure-name]]
                             {THRU   }

                {BEFORE}
    [WITH TEST {      }] UNTIL condition
                {AFTER }

    [imperative-statement END-PERFORM]

                             {THROUGH}
PERFORM [procedure-name [{       } procedure-name]]
                             {THRU   }

                {BEFORE}
    [WITH TEST {      }]
                {AFTER }

           {identifier}      {identifier}
 VARYING   {          } FROM {index-name}
           {index-name}      {literal   }
```

```
                    {identifier}
                 BY {           } UNTIL condition
                    {literal   }

             {identifier}      {identifier}
        [AFTER {          } FROM {index-name}
             {index-name}      {literal   }

                    {identifier}
                 BY {           } UNTIL condition]...
                    {literal   }

      [imperative-statement END-PERFORM]
  PURGE cd-name

                       |----------|
                    [{ | PREVIOUS | }]
                       |----------|
READ file-name-1    [{             }] RECORD [INTO identifier-1]
                    [{  NEXT       }]

      [AT END imperative-statement]

      [NOT AT END imperative-statement]

      [END-READ]

  READ file-name RECORD [INTO identifier] [KEY IS data-name]

      [INVALID KEY imperative-statement]

      [NOT INVALID KEY imperative-statement]

      [END-READ]

                   {MESSAGE}
  RECEIVE cd-name  {       } INTO identifier
                   {SEGMENT}

      [NO DATA imperative-statement]

      [WITH DATA imperative-statement]

      [END-RECEIVE]

  RELEASE record-name [FROM identifier]

  RETURN file-name RECORD [INTO identifier]

       AT END imperative-statement

      [NOT AT END imperative-statement]

      [END-RETURN]
```

```
REWRITE record-name [FROM identifier]

    [INVALID KEY imperative-statement]

    [NOT INVALID KEY imperative-statement]

    [END-REWRITE]

                              {identifier}
SEARCH identifier [VARYING    {          }]
                              {index-name}

    [AT END imperative-statement]

                    {imperative-statement}
    {WHEN CONDITION {                    }}... [END-SEARCH]
                    {NEXT SENTENCE       }

SEARCH ALL identifier

    [AT END imperative-statement]

          {          {IS EQUAL TO } {identifier           }}
          {          { |--------| } {                     }}
          {data-name { | EQUALS | } {literal              }}
          {          { |--------| } {                     }}
     WHEN {          {IS =        } {arithmetic-expression}}
          {                                                 }
          {condition-name                                   }

          {          {IS EQUAL TO } {identifier           }}
          {          { |--------| } {                     }}
          {data-name { | EQUALS | } {literal              }}
          {          { |--------| } {                     }}
    [AND  {          {IS =        } {arithmetic-expression}}]...
          {                                                 }
          {condition-name                                   }

    {imperative-statement}
    {                    } [END-SEARCH]
    {NEXT SENTENCE       }

SEND cd-name FROM identifier

                                  {WITH identifier}
SEND cd-name [FROM identifier]    {WITH ESI       }
                                  {WITH EGI       }
                                  {WITH EMI       }

                        {{identifier} [LINE ]}
    {BEFORE}            {{integer   } [LINES]}
   [{      } ADVANCING  {                    }]
    {AFTER }            { mnemonic-name      }
                        { PAGE               }

   [REPLACING LINE]
```


SET {identifier} ... TO {identifier} {index-name} {integer}
SET {index-name}... {UP BY} {identifier} {integer} {DOWN BY}
SET {SWITCH-n} {mnemonic-name} ... TO {ON} {OFF}
SET {condition-name}... TO {TRUE} {FALSE}
SET {identifier} {ADDRESS OF identifier} ... TO {identifier} {ADDRESS OF identifier} {NULL}
SORT file-name {ON {ASCENDING} {DESCENDING} KEY {data-name [FOR DATE]} }...
[WITH DUPLICATES IN {ORDER} {SEQUENCE}]
[COLLATING SEQUENCE IS {alphabet-name} {NATIVE} {STANDARD-1} {STANDARD-2} {ASCII} {EBCDIC} {GBCD} {JIS}]
{INPUT PROCEDURE IS procedure-name [{THROUGH} {THRU} procedure-name]}
{USING {file-name}...}
{OUTPUT PROCEDURE IS procedure-name [{THROUGH} {THRU} procedure-name]}
{GIVING {file-name}...}

```
|-------------------------------------------------------------|
|                       {ASCENDING }                          |
| SORT data-name-2 [ON {           } KEY [data-name-1]... ]... |
|                       {DESCENDING}                          |
|                                                             |
|                          { ORDER    }                       |
|     [WITH DUPLICATES IN {          }]                       |
|                          { SEQUENCE }                       |
|                                                             |
|                             { alphabet-name-1}              |
|                             { NATIVE         }              |
|                             { STANDARD-1     }              |
|     [COLLATING SEQUENCE IS  { STANDARD-2     } ]            |
|                             { ASCII          }              |
|                             { EBCDIC         }              |
|                             { GBCD           }              |
|                             { JIS            }              |
|-------------------------------------------------------------|
```

```
 START file-name

          {   IS EQUAL TO                 }
          {   IS =                        }
          { |---------|                   }
          { | EQUALS  |                   }
          { | EXCEEDS |                   }
          { |---------|                   }
     [KEY {   IS GREATER THAN             } data-name]
          {   IS >                        }
          {   IS NOT LESS THAN            }
          {   IS NOT <                    }
          {   IS GREATER THAN OR EQUAL TO }
          {   IS >=                       }

     [INVALID KEY imperative-statement]

     [NOT INVALID KEY imperative-statement]

     [END-START]
```

```
|-------------------------------------------------------|
|  START file-name                                      |
|                                                       |
|           {                                 }         |
|           {   IS LESS THAN                  }         |
|           {   IS <                          }         |
|       KEY {   IS NOT GREATER THAN           } data-name |
|           {   IS NOT >                      }         |
|           {   IS LESS THAN OR EQUAL TO      }         |
|           {   IS <=                         }         |
|                                                       |
|      [INVALID KEY imperative-statement]               |
|                                                       |
|      [NOT INVALID KEY imperative-statement]           |
|                                                       |
|      [END-START]                                      |
|-------------------------------------------------------|
```

```
     {   RUN     }
     {   literal }
STOP { |-------| }
     { | ERROR | }
       |-------|

        {identifier}                 {identifier}
STRING {{          }... DELIMITED BY {literal   }}...
        {literal   }                 {SIZE      }

     INTO identifier

    [WITH POINTER identifier]

    [ON OVERFLOW imperative-statement]

    [NOT ON OVERFLOW imperative-statement]

    [END-STRING]
         {identifier}
SUBTRACT {          }... FROM {identifier [ROUNDED]}...
         {literal   }

    [ON SIZE ERROR imperative-statement]

    [NOT ON SIZE ERROR imperative-statement]

    [END-SUBTRACT]

         {identifier}         {identifier}
SUBTRACT {          }... FROM {          }
         {literal   }         {literal   }

     GIVING {identifier [ROUNDED]}...

    [ON SIZE ERROR imperative-statement]

    [NOT ON SIZE ERROR imperative-statement]

    [END-SUBTRACT]

         {CORRESPONDING}
SUBTRACT {             } identifier FROM identifier [ROUNDED]
         {CORR         }

    [ON SIZE ERROR imperative-statement]

    [NOT ON SIZE ERROR imperative-statement]

    [END-SUBTRACT]

SUPPRESS PRINTING

TERMINATE {report-name}...
```

```
|-----------------------------------------------------------|
|                                                           |
| TRANSFORM identifier CHARACTERS                           |
|                                                           |
|           {figurative-constant}    {figurative-constant}  |
|      FROM {literal            } TO {literal            }  |
|           {identifier         }    {identifier         }  |
|                                                           |
|-----------------------------------------------------------|
```

```
UNSTRING identifier

                        {identifier}           {identifier}
   [DELIMITED BY [ALL] {          } [OR [ALL] {          }]... ]
                        {literal   }           {literal   }

     INTO {identifier
               [DELIMITER IN identifier]
               [COUNT IN identifier]}...

    [WITH POINTER identifier]

    [TALLYING IN identifier]

    [ON OVERFLOW imperative-statement]

    [NOT ON OVERFLOW imperative-statement]

    [END-UNSTRING]
```

```
                              {EXCEPTION}
USE [GLOBAL] AFTER STANDARD {          } PROCEDURE
                              {ERROR    }

        {{file-name}...}
        { INPUT        }
     ON { OUTPUT       }
        { I-O          }
        { EXTEND       }
```

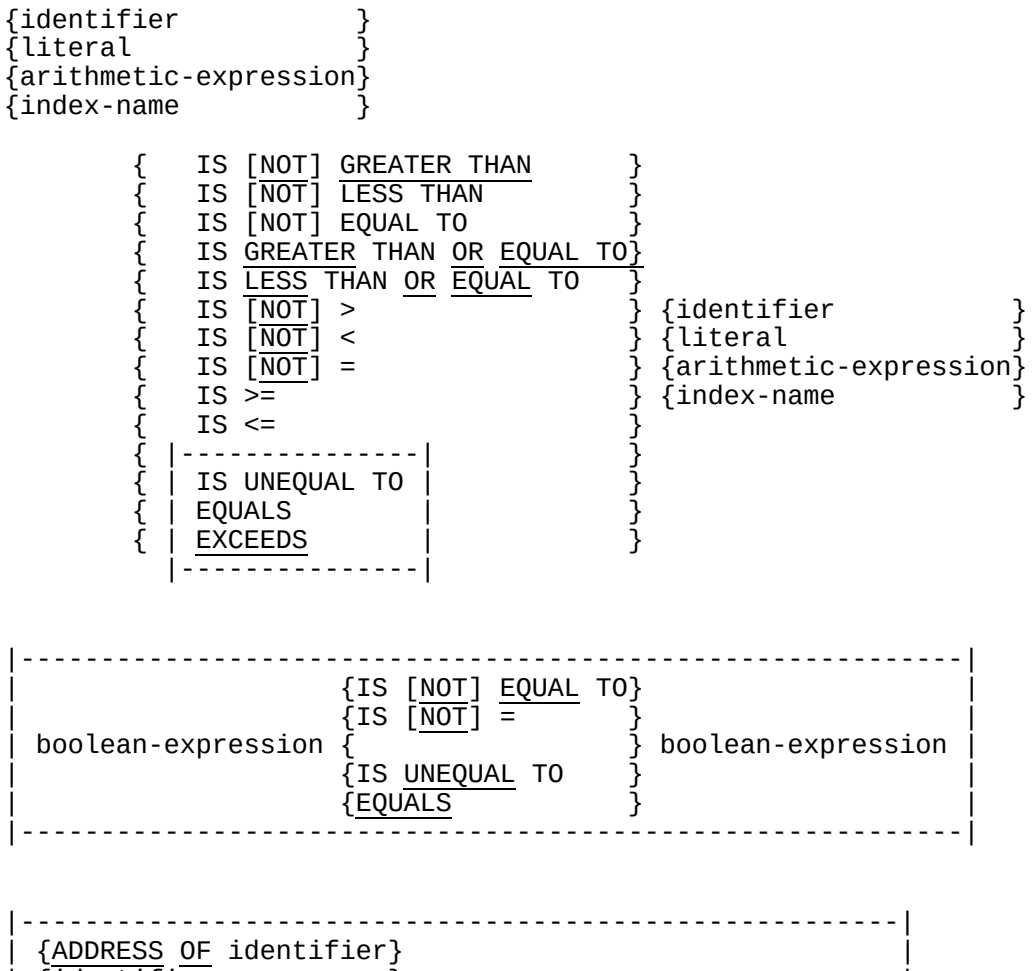
```
USE [GLOBAL] BEFORE REPORTING identifier
```

```
USE FOR DEBUGGING

        { cd-name                       }
        {[ALL REFERENCES OF] identifier}
        {        |--------------------| }
     ON {        | [WITH CONVERSION] | }...
        {        |--------------------| }
        { procedure-name                }
        { file-name                     }
        { ALL PROCEDURES                }
```

```
WRITE record-name [FROM identifier]

                          {identifier} [LINE ]
                         {{           } [     ]}
    {BEFORE}             {{integer    } [LINES]}
   [{      } ADVANCING {                       }]
    {AFTER }             {{mnemonic-name}     }
                         {{              }     }
                          {PAGE          }

        {END-OF-PAGE}
    [AT {            } imperative-statement]
        {EOP         }

            {END-OF-PAGE}
    [NOT AT {            } imperative-statement]
            {EOP         }

    [END-WRITE]

WRITE record-name [FROM identifier]

    [INVALID KEY imperative-statement]

    [NOT INVALID KEY imperative-statement]

    [END-WRITE]
```

GENERAL FORMAT FOR CONDITIONS

Relation Condition



Class Condition

```
                     {   NUMERIC          }
                     {   ALPHABETIC       }
                     {   ALPHABETIC-LOWER }
identifier IS [NOT]  {   ALPHABETIC-UPPER }
                     { |---------|        }
                     { | BOOLEAN |        }
                     { |---------|        }
                     {   class-name       }
```

Condition-name Condition

```
condition-name
```

Switch-status Condition

```
condition-name
```

Sign Condition

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

Negated Condition

```
NOT condition
```

Combined Condition

```
           {AND}
condition {{   } condition}...
           {OR }
```

Abbreviated Combined Relation Condition

```
                      {AND}
relation-condition {{   } [NOT] [relational-operator] object}...
                      {OR }
```

F.2 MISCELLANEOUS FORMATS

Qualification

Format 1

```
                    {IN}                    {IN} {file-name}
                  { {  } data-name}... [{  } {         }]}
                  { {OF}                   {OF} {cd-name  } }
{data-name     } {                                          }
{condition-name} {                                          }
                  { {IN} {file-name}                        }
                  { {  } {         }                        }
                    {OF} {cd-name  }
```

Format 2

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

Format 3

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

Format 4

```
                {IN}
LINAGE-COUNTER {  } file-name
                {OF}
```

Format 5

```
{PAGE-COUNTER} {IN}
{              } {  } report-name
{LINE-COUNTER} {OF}
```

Format 6

```
            {IN}             {IN}
           {{  } data-name [{  } report-name]}
           {{OF}            {OF}              }
data-name {                                   }
           {{IN}                              }
           {{  } report-name                  }
            {OF}
```

Subscripting

```
                    {integer                 }
                    {                        }
{data-name     }    {            {+}         }
{              } ( {data-name [{ } integer] }... )
{condition-name}    {            {-}         }
                    {                        }
                    {             {+}        }
                    {index-name [{ } integer]}
                                  {-}
```

Reference Modification

```
data-name (leftmost-character-position : [length] )
```

Identifier

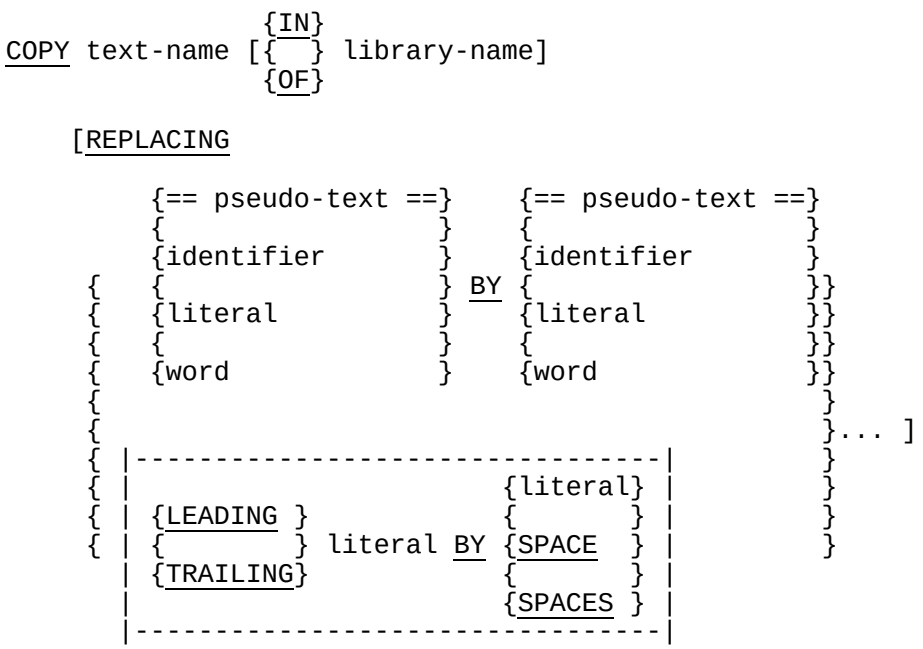
```
{                                      {file-name  }   }
{           {IN}                  {IN} {           }   }
{data-name [{  } data-name]... [{  } {cd-name    }]  }
{           {OF}                  {OF} {           }   }
{                                      {report-name}   }
{                                                      }
{         [( {subscript}... )]                         }
{                                                      }
{         [(leftmost-character-position : [length] )]}
{                                                      }
{function-identifier                                   }
```

Function-identifier

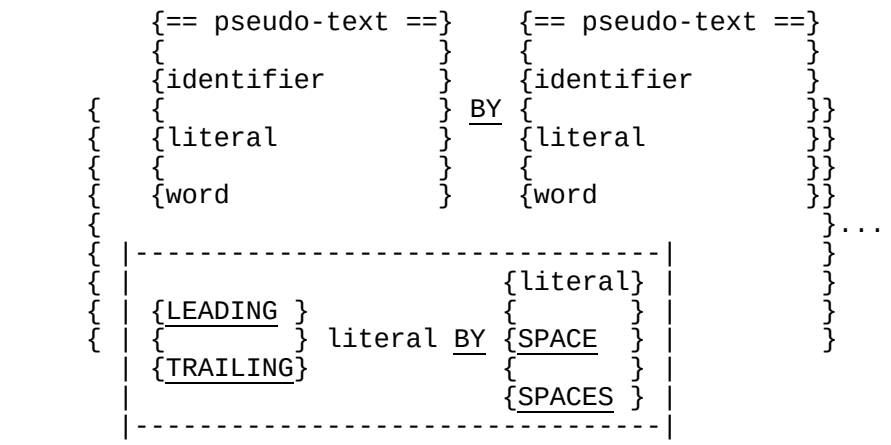
```
FUNCTION function-name [({argument}...)]
          [(leftmost-character-position:[length])]
```

F.3 GENERAL FORMAT FOR COPY AND REPLACE STATEMENTS

COPY



REPLACE



REPLACE OFF

F.4 GENERAL FORMAT FOR SEPARATELY COMPILED PROGRAM

```
[control-division]

identification-division

[environment-division]

[data-division]

[procedure-division]

[[contained-program]...

end-program-header]
```

```
|-------------------------------------------------------------|
| A contained-program is allowed anywhere a COBOL statement is |
| permitted.                                                  |
|-------------------------------------------------------------|
```

F.5 GENERAL FORMAT FOR CONTAINED-PROGRAM

```
identification-division
[environment-division]
[data-division]
[procedure-division]
[contained-program]...
end-program-header
```

F.6 GENERAL FORMAT FOR A SEQUENCE OF SEPARATELY COMPILED PROGRAMS

```
{[control-division]
identification-division
[environment-division]
[data-division]
[procedure-division]
[contained-program]...
end-program-header}...
[control-division]
identification-division
[environment-division]
[data-division]
[procedure-division]
[[contained-program]...
end-program-header]
```

Glossary

The terms in this glossary are defined in accordance with their meaning in COBOL, and may not have the same meaning for other programming languages.

abbreviated combined relation condition

The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

access mode

The manner in which records are to be operated upon within a file.

actual decimal point

The physical representation, using the decimal point character period (.) or comma (,), of the decimal point position in a data item.

|actual key

A key whose contents identify by its address on a disk a logical record in a relative file.|

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, | 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.

alphanumeric character

Any character in the computer's character set.

alphanumeric function

A function whose value is composed of a string of one or more characters from the computer's character set.

alternate record key

A key, other than the Prime Record Key, whose contents identify a record within an indexed file.

argument

An identifier, a literal or an arithmetic expression that specifies a value to be used in the evaluation of a function.

arithmetic expression

An identifier of 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 operation

The process caused by the execution of an arithmetic statement, or the evaluation of an arithmetic expression, that results in a mathematically correct solution to the arguments presented.

arithmetic operator

A single character, or a fixed two-character combination of the character(s) that belongs to the following set:

Character	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

arithmetic statement

A statement that causes an arithmetic operation to be executed. The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY , and SUBTRACT statements.

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 comparing 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:

1. During the execution of a READ statement for a sequentially accessed file when no next logical record exists for the file or when an optional file is not present.

2. During the execution of a RETURN statement, when no next logical record exists for the associated sort or merge file.

3. During the execution of a SEARCH statement, when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

block

A physical unit of data that is normally composed of one or more logical records. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either contained within the block or that overlap the block. The term is synonymous with Physical Record.

body group

Generic name for a report group of TYPE DETAIL, CONTROL HEADING or CONTROL FOOTING.

bottom margin

An empty area which follows the page body.

called program

A program which is the object of a CALL statement combined at object time with the calling program to produce a run unit.

calling program

A program which executes a CALL to another program.

cd-name

A user-defined word that names a Message Control System (MCS) interface area described in a communication description entry within the Communication Section of the Data Division.

character

The basic indivisible unit of the language.

character position

A character position is the amount of physical storage required to store a single Standard Data Format character whose usage is DISPLAY, i.e.one byte (8 bits).

character-string

A sequence of contiguous characters which form a 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 contents of an item are wholly alphabetic or wholly numeric.

class-name

A user-defined word defined in the SPECIAL-NAMES paragraph of the Environment Division that assigns a name to the proposition for which a truth value can be defined, that the content of a data item consists exclusively of those characters listed in the definition of the class-name.

clause

A clause is an ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

COBOL character set

The complete COBOL character set, exclusive of the contents of non-numeric literals, comment-entries, and comment lines, consists of the characters listed below:

Character	Meaning
0, 1, 2, 3, 4, 5, 6, 7, 8, 9	digit
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	letter (uppercase)
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,	letter (lower-case)
	space
+	plus sign
-	minus sign (hyphen)
*****	asterisk
/	stroke (virgule, slash)
=	equal sign
$ (currency symbol)	currency sign
,	comma (decimal point)
;	semi-colon
.	period (decimal point, full stop)
"	quotation mark
'	apostrophe
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol
:	colon
	'Horizontal Tabulation'
_	underscore

COBOL word

(see word)

collating sequence

The sequence in which the characters that are acceptable to a computer are ordered for purposes of sorting, merging and comparing.

column

A character position within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

combined condition

A condition that is the result of connecting two or more conditions with the AND or the OR logical operator.

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 prior to printing the comment.

comment-entry

An entry in the Identification division that may be any combination of characters from the computer character set. Comment-entries are used to document the program.

common program

A program which, despite being directly contained within another program, may be called from any other program directly or indirectly contained in that other program.

communication description entry

An entry in the Communication Section of the Data Division that is composed of the level indicator CD, followed by a cd-name, and then followed by a set of clauses as required. It describes the interface between the Message Control System (MCS) and the COBOL program.

communication device

A mechanism (hardware or hardware/software) capable of sending data to a queue and/or receiving data from a queue. This mechanism may be a computer or a peripheral device. One or more programs containing Communication Description entries and residing within the same computer define one or more of these mechanisms.

Communication Section

The section of the Data Division that describes the interface areas between the MCS and the program, composed of one or more CD description areas.

compile time

The time at which a COBOL source program is translated, by a COBOL compiler, into a COBOL object program.

compiler directing statement

A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation.

complex condition

A condition in which one or more logical operators act upon one or more conditions. (See negated simple condition, combined condition, negated combined condition).

computer-name

A system-name that identifies the computer upon which the program is to be compiled or run.

condition

A status of a program at execution time for which a truth value can be determined. Where the term 'condition' (condition-1, condition-2, ...) appears in these language specifications in or in reference to 'condition' (condition-1, condition-2, ...) of a general format, it is a conditional expression consisting or either a simple condition optionally parenthesized, or a combined condition consisting of a syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

condition-name

A user-defined word that assigns a name to a subset of values, that a conditional variable may assume; or a user-defined word assigned to a status of a switch. When 'condition-name' is used in the Formats, it represents a unique data item reference consisting of a syntactically correct combination of a condition-name and qualifiers, subscripts, and indices, as required for uniqueness of reference.

condition-name condition

The proposition, for which a truth value can be determined that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

conditional expression

A simple condition or a complex condition specified in an IF, PERFORM or SEARCH statement. (See simple condition and complex condition.)

conditional phrase

A conditional phrase specifies the action to be taken upon determination of the truth value of a condition resulting from the execution of a conditional statement.

conditional statement

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

conditional variable

A data item one or more values of which has a condition-name assigned to it.

Configuration Section

A section of the Environment Division that describes overall specifications of source and object computers.

connective

A reserved word that is used to:

1. Associate a data-name, paragraph-name, text-name, or condition-name with its qualifier.

2. Link two or more operands written in a series.

3. Form conditions (logical connectives). (See logical operator).

constant

A unit of data whose value is specified and is not subject to change.

|Constant Section

The section of the Data Division that defines the data items whose values do not change during the execution of the program.|

contained program

A contained program is a program with is contained within another COBOL program.

contiguous items

Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to each other.

control break

A change in the value of a data item that is referenced in the CONTROLS clause. More generally, a change in the value of a data item that is used to control the hierarchical structure of a report.

control break level

The relative position within a control hierarchy at which the most major control break occurred.

control data item

A data item, a change in whose contents may produce a control break.

control data-name

A data-name that appears in a CONTROL clause and refers to a control data item.

control footing

A report group that is presented at the end of the control group of which it is a member.

control group

A set of body groups that is presented for a given value of a control data item or of FINAL. Each control group may begin with a CONTROL HEADING, end with a CONTROL FOOTING, and contain DETAIL report groups.

control heading

A report group that is presented at the beginning of the control group of which it is a member.

control hierarchy

A designated sequence of report subdivisions defined by the positional order of FINAL and the data-names within a CONTROL clause.

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.

currency sign

The character "$" of the COBOL character set |, or more precisely, the character whose internal representation is the same as that of character "$" (see Appendix B).|

currency symbol

The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present in a COBOL source program, the currency symbol is identical to the currency sign.

current record

The record that is available in the record area associated with the file.

current volume pointer

A conceptual entity that points to the current volume of a sequential file.

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 item

A character or a set of contiguous characters (excluding literals in either case) defined as a unit of data by the COBOL program or by the rules for function evaluation.

data-name

A user-defined word of up to 30 characters 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 that cannot be subscripted, indexed, or qualified unless specifically permitted by the rules for that format.

debugging line

Any line with a 'D'|or a 'd'| in the Indicator Area of the line.

debugging section

A debugging section is a section that contains a USE FOR DEBUGGING statement.

declarative-sentence

A compiler-directing sentence consisting of a single USE statement terminated by the separator period.

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 keyword DECLARATIVES and the last of which is followed by the keywords END DECLARATIVES. A declarative is composed of a section header, followed by declarative sentence and optionally, one or more paragraphs.

delimited scope statement

Any statement which includes its explicit scope terminator. (See implicit and explicit scope terminators).

delimiter

A character or a sequence of contiguous characters that identify the end of a string of characters and separates that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

descending key

A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

destination

The symbolic identification of the receiver of a transmission from a queue.

de-edit

The logical removal of all editing characters from a numeric edited data item in order to determine that item's unedited numeric value.

digit position

A digit position is the amount of physical storage required to store a single digit. This amount may vary depending on the usage specified in the Data Description entry that defines the data item.

If the Data Description entry specifies that usage is DISPLAY, then a digit position is synonymous with a character position.

division

A collection of zero, one or more sections or paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. Each division consists of the division header and the related division body. There are four (4) divisions |, possibly five (5),| in a COBOL program: |Control,| Identification, Environment, Data, and Procedure.

division header

A combination of words followed by a separator period, that indicates the beginning of a division. The division headers in a COBOL program are:

```
CONTROL DIVISION.
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION [USING {data-name}... ].
```

dynamic access

An access mode in which specific logical records can be obtained from or placed into a mass storage file in a non-sequential manner and obtained from a file in a sequential manner during the scope of the same OPEN statement (see "Random Access", "Sequential Access", and "File Description" in Chapter 8).

editing character

A single character or a fixed-two-character combination, belonging to the following set:

Character	Meaning
B	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
$ (currency symbol)	currency sign
,	comma (decimal point)
.	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 COBOL source program after which no further procedures appear.

end program header

A combination of words, followed by a separator period, that indicates the end of a COBOL program. The end program header is:

```
END PROGRAM program-name.
```

entry

Any descriptive set of consecutive clauses terminated by a separator period and written in the Identification Division, Environment Division, or Data Division of a COBOL source program.

environment clause

A clause that appears as part of an Environment Division entry.

execution time

See "object time".

explicit scope terminator

A reserved word which terminates the scope of a particular Procedure Division statement.

expression

An arithmetic or conditional expression.

extend mode

The state of a file after execution of an OPEN statement, with the EXTEND phrase specified, for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file.

external data

The data described in a program as external data items and external file connectors.

external data item

A data item which is described as part of an external record in one or more programs of a run unit and which itself may be referenced from any program in which it is described.

external data record

A logical record which is described in one or more programs of a run unit and whose constituent data items may be referenced from any program in which they are described.

external file connector

A file connector which is accessible to one or more programs in the run unit.

external switch

A hardware or software device used to indicate that one of two alternate states exist.

figurative constant

A compiler-generated value referenced through the use of certain reserved words.

file

A collection of records.

file attribute conflict condition

An unsuccessful attempt has been made to execute an input-output operation on a file and the file attributes, as specified for that file in the program, do not match the fixed attributes for that file.

file clause

A clause that appears as part of any of the following Data Division entries:

File description (FD)
Sort-Merge file description (SD)

file connector

A storage area which contains information about a file and is used as the linkage between a file-name and a physical file and between a file-name and its associated record area.

FILE-CONTROL

The name of an Environment Division paragraph in which the data files for a given source program are declared.

file control entry

A SELECT clause and all its subordinate clauses which declare the relevant physical attributes of a file.

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 organization

The permanent logical file structure established at the time that a file is created.

file position indicator

A conceptual entity that contains the value of the current key within the key of reference for an indexed file, or the record number of the current record for a sequential file, or the relative record number of the current record for a relative file, or indicates that no next logical record exists, or that the number of significant digits in the relative record number is larger than the size of the relative key data item, or that an optional input file is not present, or that the at end condition already exists, or that no valid record has been established.

File Section

The section of the Data Division that contains File Description entries and Sort-Merge File Description entries together with their associated Record Descriptions.

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.

fixed file attributes

Information about a file which is established when a file is created and cannot subsequently be changed during the existence of the file.
These attributes include the organization of the file (sequential, relative, or indexed), the prime record key, the alternate record keys, the code set, the minimum and maximum record size, the record type (fixed or variable), the collating sequence of the keys for indexed files, the blocking factor, the padding character, and the record delimiter.

fixed length record

A record associated with a file whose file description or sort-merge description entry requires that all records contain the same number of character positions.

footing area

The position of the page body adjacent to the bottom margin.

format

A specific arrangement of a set of data.

function

A temporary data item whose value is determined by invoking a mechanism provided by the compiler at the time the function is referenced during the execution of a statement.

function-identifier

A syntactically correct combination of character-strings and separators that references a function. The data item represented by a function is uniquely identified by a function-name with its arguments, if any. A function-identifier may include a reference-modifier. A function identifier that references an alphanumeric function may be specified anywhere in the general formats that an identifier may be specified, subject to certain restrictions. A function-identifier that references an integer or numeric function may be referenced anywhere in the general formats that an arithmetic expression may be specified.

function-name

A word that names a mechanism provided by the compiler to determine the value of a function.

global name

A name which is declared in only one program but which may be referenced from that program and from any program contained within that program. Condition-names, data-names, file-names, record-names, report-names, and some special registers may be global names. (See Chapter 3).

group item

A data item that is composed of subordinate data items.

high-order end

The leftmost character of a string of characters.

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 without the REEL or UNIT phrase for that file.

I-O status

A conceptual entity which contains the two-character value indicating the resulting status of an input-output operation. This value is made available to the program through the use of the FILE STATUS clause in the file control entry for the file.

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-CONTROL entry

An entry in the I-O-CONTROL paragraph of the Environment Division which contains clauses which provide information required for the transmission and handling of data on named files during the execution of a program.

identifier

A syntactically correct combination of character-strings and separators that names a data item. When referencing a data item which is not a function, an identifier consists of a data-name, together with its qualifiers, subscripts, and reference-modifier, as required for uniqueness of reference. When referencing a data item which is a function, a function-identifier is used. The rules for 'identifier' associated with general Formats may, however, specifically prohibit reference to functions, qualification, subscripting or reference modification.

imperative statement

A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement can consist of a sequence of imperative statements.

implicit scope terminator

A separator period which terminates the scope of any preceding unterminated statement, or a phrase of a statement which by its occurrence indicates the end of the scope of any statement contained within the preceding phrase.

index

A computer storage area 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 values associated with an index-name can be stored in a form specified by the system.

index-name

A user-defined word that names an index associated with a specific table.

indexed data-name

An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

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.

initial program

A program that is placed into an initial state every time the program is called in a run unit.

initial state

The state of a program when it is first called in a run unit.

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 without the REEL or UNIT phrase for that file.

input procedure

A set of statements that is executed each time a record is released to the sort 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 required by an object program and which provides information required for transmission and handling of data during execution the object program.

input-output statement

A statement that causes files to be processed by performing operations upon individual records or upon the file as a unit. The input-output statements are: ACCEPT (with the identifier phrase), CLOSE, DELETE, DISABLE, DISPLAY, ENABLE, OPEN, PURGE, READ, RECEIVE, REWRITE, SEND, SET (with the TO ON or TO OFF phrase), START, and WRITE.

integer

(1) A numeric literal that does not include any digit positions to the right of the decimal point.

(2) A numeric data item defined in the Data Division that does not include any digit positions to the right of the decimal point.

(3) A numeric function whose definition provides that all digits to the right of the decimal point are zero in the returned value for any possible evaluation of the function.

Where the term 'integer' appears in the general formats, integer must be a numeric literal which is an integer, and it must be neither signed nor zero unless explicitly allowed by the rules for that format.

Integer Function

A function whose category is numeric and whose definition provides that all digits to the right of the decimal point are zero in the returned value for any possible evaluation of the function.

internal data

The data described in a program excluding all external data items and external file connectors. Items described in the Linkage Section of a program are treated as internal data.

internal data item

A data item which is described in one program in a run unit. An internal data item may have a global name.

internal file connector

A file connector which is accessible to only one object program in the run unit.

intra-record data structure

The entire collection of groups and elementary data items from a logical record which is defined by a contiguous subset of the data description entries which describe that record. These data description entries include all entries whose level-number is greater than the level-number of the first data description entry describing the intra-record data structure.

invalid key condition

A condition, at object time, caused when a specific value of the key associated with an indexed or relative file is determined to be invalid.

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, either prime or alternate, currently being used to access records within an indexed file.

keyword

A reserved word or a function-name whose presence is required when the format in which the word appears is used in a source program.

language-name

A system-name that specifies a particular programming language.

letter

A character belonging to one of the following two sets:

(1) uppercase 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;`

(2) lower-case 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.`

level indicator

Two alphabetic characters that identify a specific type of file or a position in a hierarchy.

level-number

A user-defined word, expressed as a one or two-digit number, which indicates the hierarchical position of a data item or the special properties of a Data Description entry. Level-numbers in the range 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-numbers 66, 77, and 88 identify special properties of a Data Description entry.

library text

A sequence of character-strings and/or separators in a COBOL library.

library-name

A user-defined word that names a COBOL library that is to be used by the compiler for a given source program compilation.

LINAGE-COUNTER

A special register whose value points to the current position within the page body.

line

(See Report Line)

line number

An integer that denotes the vertical position of a report line on a page.

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 the called programs.

literal

A character-string whose value is implied by the ordered set of characters comprising the string.

logical operator

One of the reserved words AND, OR, or NOT. In the formation of a condition, either AND or OR, or both, can be used as logical connectives. NOT can be used for logical negation.

logical page

A conceptual entity consisting of the top margin, the page body, and the bottom margin.

logical record

The most inclusive data item. The level-number for a record is 01. A record may be either an elementary item or a group item.

low-order end

The rightmost character of a string of characters.

mass storage

A storage medium in which data can be organized and maintained in both a sequential and non-sequential manner.

Mass Storage Control System (MSCS)

An input-output control system that directs, or controls, the processing of mass storage files.

mass storage file

A collection of records that is assigned to a mass storage medium.

MCS

See Message Control System.

merge file

A collection of records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

message

Data associated with an End of Message Indicator or an End of Group Indicator. (See Message Indicators)

Message Control System (MCS)

A DPS 7 supplied subsystem that supports the processing of messages.

Message Count

The count of the number of complete messages that exist in the designated queue of messages.

message indicators

EGI (End of Group Indicator), EMI (End of Message Indicator), and ESI (End of Segment Indicator) are conceptual indications that serve to notify the MCS that a specific condition exists (end of group, end of message, end of segment). Within the hierarchy of EGI, EMI, and ESI, an EGI is conceptually equivalent to an ESI, EMI, and EGI.

An EMI is conceptually equivalent to an ESI and EMI. Thus, a segment may be terminated by a ESI, EMI, or EGI. A message may be terminated by an EMI or EGI.

message segment

Data that forms a logical subdivision of a message, normally associated with an end of segment indicator. (See Message Indicators)

mnemonic-name

A user-defined word that is associated in the Environment Division with a specific implementor-name.

native character set

The character set associated with the DPS 7 computer i.e. the EBCDIC character set.

native collating sequence

The EBCDIC collating sequence.

negated combined condition

The 'NOT' logical operator immediately followed by a parenthesized combined condition.

negated simple condition

The 0'NOT' logical operator immediately followed by a simple condition.

next executable sentence

The next sentence to which control is to 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 that logically follows the current record of a file.

next record pointer

A conceptual entity that points to the next logical record, indicates the at end condition, or is set to indicate that no valid next record has been established.

non-contiguous items

Elementary data items, in the Working-Storage, |Constant| and Linkage Sections, that bear no hierarchic relationship to other data items.

non-numeric 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 non-numeric items may be formed from more restricted character sets.

non-numeric literal

A literal bounded by quotation marks |or apostrophes.| The string of characters can include any character in the computer's character set, |some or all of which may be represented by a symbolic-character string.|

numeric-character

A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

numeric function

A function whose class and category are numeric but which for some possible evaluation does not satisfy the requirements of an integer function.

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 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 entry

An entry in the OBJECT-COMPUTER paragraph of the Environment Division which contains clauses which describe the computer environment in which the object program is to be executed.

object of entry

A set of operands and reserved words, within a Data Division entry, that immediately follows the subject of the entry.

object program

An object program is the machine language result of the operation of a COBOL compiler on a source program.

object time

The time at which an object program is executed.

OBJECT-COMPUTER

The name of an Environment Division paragraph that describes the computer environment within which the object program is executed.

obsolete element

A COBOL language element in Standard COBOL that is to be deleted from the next revision of Standard COBOL.

open mode

The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file. The particular open mode specified in the OPEN statement must be one of the following: 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 manual, any lower-case 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 file

A file which is declared as being not necessarily present each time the object program is executed. The object program causes an interrogation for the presence or absence of the file.

optional word

A reserved word that is included in a specific 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 without the REEL or UNIT phrase for that file.

output procedure

A set of statements to which control is given during the execution of a SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function has selected the next record in merge order.

padding character

An alphanumeric character used to fill the unused character positions in a physical record.

page

A vertical division of a report representing a physical separation of report data, the separation being based on internal reporting requirements and/or external characteristics of the reporting medium.

page body

That part of the logical page in which lines can be written and/or spaced.

page footing

A report group that is presented at the end of a report page as determined by the Report Writer Control System.

page heading

A report group that is presented at the beginning of a report page as determined by the Report Writer Control System.

paragraph

In the Procedure Division, a paragraph-name followed by a separator period and by zero, 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 the separator period, 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 COBOL procedural statement or of a COBOL clause.

physical page

A device dependent concept.

physical record

(See block)

previous record

The record that logically precedes the current record of a file.

prime record key

A key whose contents uniquely identify a record within an indexed file.

printable group

A report group that contains at least one print line.

printable item

A data item, the extent and contents of which are specified by an elementary report entry. This elementary report entry contains a COLUMN NUMBER clause, a PICTURE clause, and a SOURCE, SUM or VALUE clause.

procedure

A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

procedure branching statement

A statement that causes the explicit transfer of control to a statement other than the next executable statement in the sequence in which the statements are written in the source program. The procedure branching statements are: ALTER, CALL, EXIT, EXIT PROGRAM, GO TO, MERGE (with the OUTPUT PROCEDURE phrase), PERFORM and SORT (with the INPUT PROCEDURE or OUTPUT PROCEDURE phrase).

procedure-name

A user-defined word that is used to name a paragraph or section in the Procedure Division. It consists of a paragraph-name (which can be qualified), or a section-name.

program identification entry

An entry in the PROGRAM-ID paragraph of the Identification Division which contains clauses that specify the program-name and assign selected program attributes to the program.

program-name

A user-defined word that identifies a COBOL source program.

pseudo-text

A sequence of character-strings and/or separators bounded by, but not including, pseudo-text delimiters.

pseudo-text delimiter

Two contiguous equal sign (=) characters used to delimit pseudo-text.

punctuation character

A character that belongs to the following set:

Character	Meaning
,	comma
;	semi-colon
.	period (full stop)
"	quotation mark
|'	apostrophe|
(	left parenthesis
)	right parenthesis
	space
	|'Horizontal Tabulation'|
=	equal sign

qualified data-name

An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF or IN followed by a data-name qualifier.

qualifier

1. A data-name that is used in a reference together with another data-name at a lower level in the same hierarchy.

2. A section-name that is used in a reference together with a paragraph-name specified in that section.

3. A library-name that is used in a reference together with a text-name that is part of that library.

queue

A logical collection of messages awaiting transmission or processing.

queue name

A symbolic name that indicates to the (MCS) the logical path by which a message may be accessible in a queue.

random access

An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from or placed into a relative or indexed file.

record

(See logical record)

record area

A storage area allocated for the purpose of processing the record described in a Record Description entry in the File Section of the Data Division.

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 whose contents identify a record within an indexed file.

Within an indexed file, a record key is either the Prime Record Key or an Alternate Record Key.

record-name

A user-defined word that names a record described in a Record Description entry in the Data Division of a COBOL program.

record number

The ordinal number of a record in the file whose organization is sequential.

reel

A discrete portion of a storage medium that contains part of a file, all of a file, or any number of files. The term is synonymous with unit and volume.

reference format

A format that provides a standard method for describing COBOL source programs.

reference modifier

A syntactically correct combination of character-strings and separators the defines a unique data item. It includes a delimiting left parenthesis separator, the leftmost character position, a colon separator, optionally a length, and a delimiting right parenthesis separator.

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 specific 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 meanings are:

Relational Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] > |EXCEEDS|	Greater than or not greater than
IS [NOT] LESS THAN IS [NOT] <	Less than or not less than
IS [NOT] EQUAL TO IS [NOT] = |IS UNEQUAL TO EQUALS|	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.

relative record number

The ordinal number of a record in a file whose organization is relative. This number is treated as a numeric literal which is an integer.

report clause

A clause, in the Report Section of the Data Division, that appears in a Report Description entry or a Report Group Description entry.

report description entry

An entry in the Report Section of the Data Division that is composed of the level indicator RD, followed by a report name, followed by a set of report clauses as required.

report file

An output file whose File Description entry contains a REPORT clause. The contents of a report file consist of records that are written under control of the Report Writer Control System.

report footing

A report group that is presented only at the end of a report.

report group

In the Report Section of the Data Division, an 01 level-number entry and its subordinate entries.

report group description entry

An entry in the Report Section of the Data Division that is composed of the level-number 01, the optional data-name, a TYPE clause, and an optional set of report clauses.

report heading

A report group that is presented only at the beginning of a report.

report line

A division of a page representing one row of horizontal character positions. Each character position of a report line is aligned vertically beneath the corresponding character position of the report line above it. Report lines are numbered from 1, by 1, starting at the top of the page.

Report Section

The section of the Data Division that contains one or more Report Description entries and their associated Report Group Description entries.

report writer control system (RWCS)

An object time control system, that accomplishes the construction of reports.

report writer logical record

A record that consists of the Report Writer print line and associated control information necessary for its selection and vertical positioning.

report-name

A user-defined word that names a report described in a Report Description entry within the Report Section of the Data Division.

reserved word

A COBOL word specified in the list of words that may be used in a COBOL source program, but which must not appear in the programs as user-defined words or system-names.

resource

A facility or service, controlled by the operating system, that can be used by an executing program.

resultant identifier

A user-defined data item that is to contain the result of an arithmetic operation.

routine-name

A user-defined word that identifies a procedure written in a language other than COBOL.

run unit

An object program during execution.

RWCS

(See report writer control system)

section

A set of zero, 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 separator period that indicates the beginning of a Section in |the Control,| Environment, Data, and Procedure Divisions.

In the |Control,| Environment and Data Division, a section header is composed of reserved words followed by a separator period. The permissible section headers are:

In the Control Division

SUBSTITUTION SECTION.
DEFAULT SECTION.

In the Environment Division

CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.

In the Data Division

FILE SECTION.
WORKING-STORAGE SECTION.
CONSTANT SECTION.
LINKAGE SECTION.
COMMUNICATION SECTION.
REPORT 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 separator period.

section-name

A user-defined word that names a section in the Procedure Division.

segment-number

A user-defined word that classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers may contain only the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. A segment-number may be expressed as either a one or two-digit number.

sentence

A sequence of one or more statements, the last of which is terminated by a separator period.

separately compiled program

A program which, together with its contained programs, is compiled separately from all other programs.

separator

A character or two contiguous characters 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
condition-name condition
switch-status condition
sign condition

sort file

A collection of records to be sorted by a SORT statement. The sort file is created and can be used by the sort function only.

sort-merge file description entry

An entry in the File Section of the Data Division that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

source

The symbolic identification of the originator of a transmission to a queue.

source computer entry

An entry in the SOURCE-COMPUTER paragraph of the Environment Division which contains clauses which describe the computer environment in which the source program is to be compiled.

source item

An identifier designated by a SOURCE clause that provides the value of a printable item.

source program

Although it is recognized that a source program may be represented by other forms and symbols, in this manual it always refers to a syntactically correct set of COBOL statements. A COBOL source program commences with |a Control Division or| an Identification Division and terminates with the end of the Procedure Division. In contexts where there is no possibility of ambiguity, the word 'program' alone may replace the phrase 'source program'.

SOURCE-COMPUTER

The name of an Environment Division paragraph in which the computer environment, within which the source program is compiled, is described.

special character

A character that belongs to the following set:

Character	Meaning
+	plus sign
-	minus sign
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$ (currency symbol)	currency sign
,	comma (decimal point)
;	semi-colon
.	period (decimal point)
"	quotation mark
|'	apostrophe|
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol
:	colon
|_	'Horizontal Tabulation'|
|_	underscore|

special names entry

An entry in the SPECIAL-NAMES paragraph of the Environment Division which provides means for specifying the current sign; choosing the decimal point; specifying symbolic characters; relating implementor-names to user-specified mnemonic-names; relating alphabet-names to character sets or collating sequences; and relating class-names to sets of characters.

special registers

Compiler generated storage areas whose primary use is to store information produced in conjunction with the use of specific COBOL features.

special-character word

A reserved word that 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.

standard data format

The concept used in describing 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.

sub-queue

A logical hierarchical division of a queue.

subject of entry

An operand or reserved word that appears immediately following the level indicator or the level-number in a Data Division entry.

subprogram

(See Called Program).

subscript

An occurrence number represented by either an integer, a data-name optionally followed by an integer with the operator + or -, an index-name optionally followed by an integer with the operator + or -, |or an arithmetic expression enclosed with parenthesis| that identifies a particular element in a table. A subscript may be the word ALL when the subscripted identifier is used as a function argument.

subscripted data-name

An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

sum counter

A signed numeric data item established by a SUM clause in the Report Section of the Data Division. The sum counter is used by the Report Writer Control System to contain the result of designated summing operations that take place during production of a report.

switch-status condition

The proposition, for which a truth value can be determined, that a JCL switch, capable of being set to an 'on' or 'off' status, has been set to a specific status.

symbolic-character

A group of from one (1) to thirty (30) characters combined from the set of 'O', '1', ..., '9'. A symbolic-character is used in a non-numeric literal to represent a specific character in a particular character set.

symbolic-character-string

|A symbolic-character or a group of symbolic-characters that appear within a non-numeric literal enclosed in quotation marks (or apostrophes depending on the non-numeric literal delimiter) and separated from each other by either the separator comma or space. Each symbolic-character represents a character within a given character set.|

system-name

A COBOL word which is used to communicate with the operating environment.

table

A set of logically consecutive items of data that are defined in the Data Division of a COBOL program by means of the OCCURS clause.

table element

A data item that belongs to the set of repeated items comprising a table.

terminal

The originator of a transmission to a queue, or the receiver of a transmission from a queue.

text-name

A user-defined word that identifies library text.

text-word

A character or a sequence of contiguous characters between margin A and margin R in a COBOL library, source program, or in pseudo-text which is:

1. A separator, except for: space; a pseudo-text delimiter; and the opening and closing delimiters for non-numeric literals. The right parenthesis and left parenthesis characters, regardless of context within the library, source program, or pseudo-text, are always considered text-words.

2. A literal including, in the case of non-numeric literals, the opening quotation mark which bound the literal.

3. Any other sequence of contiguous COBOL characters except comment lines and the word 'COPY', bounded by separators, which is neither a separator nor a literal.

top margin

An empty area which precedes the page body.

truth value

The representation of the result of the evaluation of a condition in terms of one of two values: true or false.

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 by + 1 or - 1 respectively.

unit

The definition of a unit depends on the number of files (units) assigned to the COBOL file:

1. If only one file has been assigned to the COBOL file, the unit is the physical unit (tape reel, disk unit etc.)

2. If more than one file has been assigned to the COBOL file each file constitutes a logical unit.

unsuccessful execution

The attempted execution of a statement that does not result in the execution of all the operations specified by that statement. The unsuccessful execution of a statement does not affect any data referenced by that statement, but may affect status indicators.

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 can be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

variable length record

A record associated with a file whose file description or sort-merge description entry permits records to contain a varying number of character positions.

variable occurrence data item

A variable occurrence data item is a table element which is repeated a variable number of times. Such an item must contain an OCCURS DEPENDING ON clause in its data description entry, or be subordinate to such an item.

variable-length data item

|A variable-length data item is a data item which, although physically fixed in size, contains a logically variable number of characters. Such an item must contain the PICTURE symbol 'L' in its Data Description entry.|

verb

A word that expresses an action to be taken by a COBOL compiler or object program.

volume

A discrete portion of a storage medium that contains part of a file, all of a file, or any number of files.

word

A character-string of not more than 30 characters which forms a user-defined word, a system-name, a reserved word or a function-name.

Working-Storage Section

The section of the Data Division that describes Working-Storage data items and constants, composed of non-contiguous items or of Working-Storage records, or of both.

77-level-description-entry

A data description entry that describes a non-contiguous data item with the level-number 77.

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Technical publication remarks form

Title :	DPS7000/XTA NOVASCALE 7000 COBOL85 Reference Manual Languages: COBOL

Reference N° :	47 A2 05UL 04	**Date:**	**November 1997**

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please include your complete mailing address below.

NAME : _____ Date : _____

COMPANY : _____

ADDRESS : _____

_____

Please give this technical publication remarks form to your BULL representative or mail to:

Bull - Documentation Dept.
1 Rue de Provence
BP 208
38432 ECHIROLLES CEDEX
FRANCE
info@frec.bull.fr

Technical publications ordering form

To order additional publications, please fill in a copy of this form and send it via mail to:

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

Phone: +33 (0) 2 41 73 72 66
FAX: +33 (0) 2 41 73 70 66
E-Mail: srv.Duplicopy@bull.net

CEDOC Reference #	Designation	Qty
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
[_ _] : The latest revision will be provided if no revision number is given.		

NAME: ______________________________ Date: __________

COMPANY: ______________________________

ADDRESS: ______________________________

____

PHONE: ______________________ FAX: ______________

E-MAIL: ______________________________

For Bull Subsidiaries:

Identification: ______________________________

For Bull Affiliated Customers:

Customer Code: ______________________________

For Bull Internal Customers:

Budgetary Section: ______________________________

For Others: Please ask your Bull representative.

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
47 A2 05UL 04