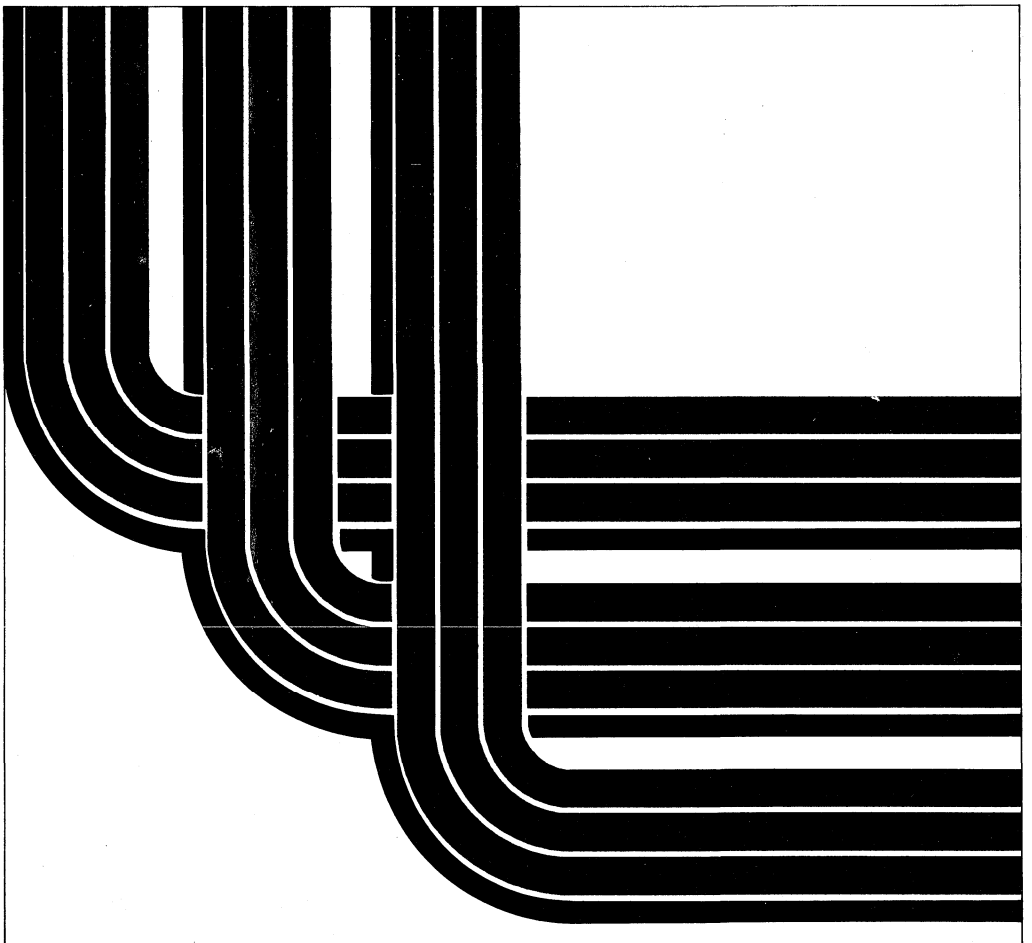
Application System/400

SC09-1380-01

Languages:
Systems Application Architecture AD/Cycle COBOL/400 Reference

Version 2



Application Development

Application System/400

SC09-1380-01

Languages:
Systems Application Architecture AD/Cycle COBOL/400 Reference

Version 2

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 1.

Second Edition (September 1992)

This edition applies to the IBM Systems Application Architecture* AD/Cycle* COBOL/400* licensed program (Program 5738-CB1), Version 2 Release 2, and to all subsequent releases and modifications until otherwise indicated in new editions. This major revision makes obsolete SC09-1380-00. Make sure you are using the proper edition for the level of the product.

Order publications through your IBM representative or the IBM branch serving your locality. Publications are not stocked at the address given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, you can address your comments to:

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 Don Mills Road
North York, Ontario, Canada M3C 1V7

You can also send your comments by facsimile to (416) 448-6057, addressed to the attention of the RCF Coordinator. If you have access to Internet, you can send your comments electronically to **torrcf@vnet.ibm.com**; IBMlink*, to **toribm(torrcf)**; IBM PROFS*, to **torolab4(torrcf)**.

If you choose to respond through Internet, please include either your entire Internet network address, or a postal address.

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you or restricting your use of it.

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

This publication could contain technical inaccuracies or typographical errors.

This publication may refer to products that are announced but not currently available in your country. This publication may also refer to products that have not been announced in your country. IBM makes no commitment to make available any unannounced products referred to herein. The final decision to announce any product is based on IBM's business and technical judgment.

Important changes or additions to the text are indicated by a vertical line (|) to the left of the change or addition. Refer to the "Summary of Changes" on page 3 for a summary of recent changes to the COBOL/400 licensed program.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

This publication contains small programs that are furnished by IBM as simple examples to provide an illustration. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. All programs contained herein are provided to you "AS IS". THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED.

Programming Interface Information

This manual is intended to help you write COBOL/400* programs. It contains information necessary for you to use the COBOL/400 compiler. This manual documents no programming interfaces for use in writing programs that request or receive the services of the COBOL/400 compiler.

Trademarks and Service Marks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

Application System/400	AD/Cycle	AS/400
CICS/400	COBOL/2	COBOL/400
IBM	IBMlink	Operating System/400
OS/2	OS/400	PROFS
Systems Application Architecture	SAA	400

The following term, denoted by a double asterisk (**) in this publication, is a trademark of another company as follows:

FLOW-MATIC	Sperry Rand Corporation

Summary of Changes

Version 2 Release 1 Modification 1 Changes

The following is a summary of changes that occurred in the Version 2 Release 1 Modification 1 COBOL/400 compiler. This information is also documented in the *Version 2 Release 1.1 Changes* manual.

TGTRLS (Target Release) Parameter: The TGTRLS (target release) parameter of the CRTCBLPGM command now has a new *release-level* value that allows you to specify the operating system release level running on the target system on which a program is to be used.

Improvements to Extended ACCEPT and Extended DISPLAY Operations: The EXTDSPOPT (Extended ACCEPT and DISPLAY Options) parameter is now available in the CRTCBLPGM command and the PROCESS statement.

This parameter provides two options: *NOUNDSPCHR that allows you to specify the type of data that extended ACCEPT and extended DISPLAY statements handle in a particular program, and *DFRWRT that allows you to specify if you want to buffer the processing of extended DISPLAY statements.

You can use the EXTDSPOPT parameter only when you specify the *EXTACCDSP generation option, or the EXTACCDSP PROCESS statement option.

Although the deferred writing (*DFRWRT) option improves performance by buffering data streams generated by consecutive extended DISPLAY statements, the no deferred writing (*NODFRWRT) option allows you to associate data errors with the statements that cause them by performing extended DISPLAY statements as they are encountered.

The no undisplayable characters (*NOUNDSPCHR) option allows you to use extended ACCEPT and extended DISPLAY statements on local workstations, or on display stations attached to remote 3174 and 3274 controllers, provided that the data does not contain undisplayable characters.

In previous releases and modifications, extended ACCEPT and extended DISPLAY statements were processed as if the default EXTDSPOPT options (*DFRWRT and *UNDSPCHR) were in effect.

SAA Data Types: The COBOL/400 compiler allows you to convert three SAA data types from externally described files into standard COBOL data items. The SAA data types that you can convert are date, time, and timestamp. These conversions are available through the *DATETIME option of the CRTCBLPGM CVTOPT parameter, or the DATETIME option of the PROCESS statement.

Variable-length Fields: You can bring a variable-length field into your program if you specify the *VARCHAR option of the CRTCBLPGM CVTOPT parameter, or the VARCHAR option of the PROCESS statement.

Null-capable Fields: Although your program can process null-capable fields, null values are not supported. When a file with one or more null-capable fields is opened successfully, file status 0P is returned. If you try to perform READ, SORT, or MERGE operations using fields that contain null values, errors occur.

Time-separation Characters: The TIMSEP parameter of job-related commands (such as CHGJOB) now specifies the time-separation character used in the WHEN-COMPILED special register, and in the time stamps that appear on compiler listings.

LIKE Clause: When you use the BLANK WHEN ZERO attribute, the PICTURE IS portion of the generated comment is now shown in a concise format, even if the parent item is numeric-edited.

Version 2 Release 2 Changes

The following is a summary of changes that occurred in Version 2 Release 2 of the COBOL/400 compiler.

Saving Data While Programs are Active: Application programs that change objects or data may run while the objects or data are being saved. Refer to the *Advanced Backup and Recovery Guide*, SC41-8079, for possible programming considerations related to Save While Active support.

LENGTH OF Special Register: You can make your programs easier to maintain with the new LENGTH OF special register, by using LENGTH OF instead of a specific length for data items. The LENGTH OF special register returns the length of a data item in number of bytes, and can be specified in the procedure division where a numeric literal is allowed.

Declaring Pointer Data Items: You can now declare and use pointer data items by specifying a usage of POINTER. Pointer data items can be set, compared for equality, and passed to other programs. The addition of this support allows easier migration of COBOL code from other SAA platforms, and allows access to Application Programming Interfaces (APIs) and languages that require pointers.

ADDRESS OF Special Register and Null Figurative Constant: Pointer data items allow the use of the ADDRESS OF special register and the NULL figurative constant. The ADDRESS OF special register returns a pointer containing the address of an item in the linkage section of a program. The NULL figurative constant returns a pointer whose address is not valid. Like pointers, the ADDRESS OF special register can be set, compared for equality, and passed to other programs.

Addition to CALL USING Statement: The CALL USING statement can now contain the ANSI 85 high-level phrases BY CONTENT and BY REFERENCE. The BY REFERENCE phrase is the default for the current CALL statement. The BY CONTENT phrase is similar to the BY REFERENCE phrase, except that copies of the USING data items are passed so that the original values in the calling program cannot be changed by the called program.

Error-Handling and Multiple Run Unit APIs: Three new APIs have been defined for use with COBOL/400 programs:

QLRCHGCM - Change COBOL Main Program allows the definition of multiple run units

QLRRTVCE - allows you to retrieve the name of the current or pending error-handling routine
QLRSETCE - allows you to specify the identity of a COBOL error-handling program for a run unit.

Together, these APIs allow user-supplied error handlers to be specified for each run unit. These APIs are documented in the *System Programmer's Interface Reference*, SC41-8223.

MOVE with DE-EDITING and CALL (NOT) ON EXCEPTION: Two new ANSI 85 high-level functions are now available: MOVE WITH DE-EDITING and CALL (NOT) ON EXCEPTION. MOVE WITH DE-EDITING adds function and usability to the MOVE statement by allowing a numeric-edited item to be moved into a numeric or a numeric-edited item. The ON EXCEPTION phrase has been added to the CALL statement so that when an error occurs on a subprogram call, the imperative statement following the ON EXCEPTION is processed. Similarly, the NOT ON EXCEPTION is processed if no errors occur when a subprogram is called.

Improving Run-Time Performance: Run-time performance can be improved by the following enhancements:

Compiler options *NOSTDINZ and *STDINZ allow you to selectively turn off or turn on the automatic initialization of user-declared data structures to machine defaults.

Sequential blocking is now done for dynamically-accessed files that are open for input or output, and for sequentially-accessed files that are open for input and subject to a START statement. This sequential blocking can be conditionally turned off or on with the *NOBLK and *BLK compiler options.

Long Key Support: The maximum key size for data base files changes from 120 to 2 000 bytes. For more information, refer to the *DDS Reference*.

Limited Support for DBCS-Graphic Data Type: The COBOL/400 compiler now provides limited support for the new Double Byte Character Set (DBCS) graphic data type from files. The compiler allows the DBCS-graphic field to be brought in as a character field in a program. Two new values are added to the CVTOPT parameter of the CRTCBLPGM command: *GRAPHIC and *NOGRAPHIC. Also, the equivalent PROCESS statement options are available: CVTGRAPHIC and NOCVTGRAPHIC.

Customer Information Control System (CICS) Support: COBOL/400 applications can now use the facilities of the CICS/400* product.

Extended ACCEPT/DISPLAY Statement: The SIZE clause is now supported on the extended DISPLAY statement. Two new options, *ACCUPDALL and *ACCUPDNE, are now available on the EXTDSPOPT parameter of the CRTCBLPGM command. These options affect the predisplaying of items in extended ACCEPT operations.

Application Development Manager/400 Support: You can now use the Application Development Manager/400 product with the COBOL/400 compiler to help you manage multiple versions of program source and program objects in an application. It also provides an application build facility.

AD/Cycle CODE/400* Support: New options and parameters have been added to the CRTCBLPGM command to support the use of the CODE/400 product to edit, compile and debug COBOL/400 programs on programmable workstations.

About This Manual

This manual describes the COBOL/400 language structure, program organization, procedure division statements, and compiler-directing statements.

This manual might refer to products that are announced but are not yet available.

The COBOL/400 language delivers most elements of the IBM Systems Application Architecture* (SAA*) Common Programming Interface (CPI) COBOL, and is the implementing product on the AS/400* system.

Who Should Use this Manual

This manual is a guide for the COBOL/400 programming language on the AS/400 system. It is intended for people who have a basic understanding of data processing concepts and of the COBOL programming language.

Before you use this manual, you should be familiar with certain AS/400 system information:

- For a comprehensive breakdown of information that pertains to COBOL/400 concepts, use the *Publications Guide, GC41-9678*, to locate reference material.
- You should be familiar with your display station (also known as a work station), and its controls. There are also some elements of its display and certain keys on the keyboard that are standard regardless of which software system is currently running at the display station, or which hardware system the display station is hooked up to. Some of these keys are:
  - Cursor movement keys
  - Command keys
  - Field exit keys
  - Insert and delete keys
  - The Error Reset key.

  This information is contained in *New User's Guide*, SC41-8211.
- You should know how to operate your display station when it is hooked up to the IBM AS/400 system and running AS/400 software. This means knowing about the OS/400* operating system and the Control Language (CL) to do such things as:
  - Sign on and sign off the display station
  - Interact with displays
  - Use Help
  - Enter control commands
  - Call utilities
  - Respond to messages.

  To find out more about this operating system and its control language, refer to:
  - *Programming: Control Language Reference*, SC41-0030
  - *Programming: Control Language Programmer's Guide*, SC41-8077

- The *Data Management Guide*, SC41-9658, provides information on using data management support, which allows an application to work with files.

  The manual includes information on:

  - Fundamental structure and concepts of data management support on the system
  - Data management support for display stations, printers, tapes, and diskettes, as well as spooling support
  - Overrides and file redirection (temporarily making changes to files when an application is run)
  - Copying files by using system commands to copy data from one place to another
  - Tailoring a system using double-byte data.

- You should know how to call and use certain utilities available on the AS/400 system:

  - The Screen Design Aid (SDA) utility used to design and code displays. This information is contained in *Application Development Tools: Screen Design Aid User's Guide and Reference*, SC09-1340.
  - The Source Entry Utility (SEU), which is a full-screen editor you can use to enter and update your source and procedure members. This information is contained in *Application Development Tools: Source Entry Utility User's Guide and Reference*, SC09-1338.

- You should know how to interpret displayed and printed messages. This information is contained in the *Languages: Systems Application Architecture* AD/Cycle* COBOL/400* User's Guide*, SC09-1383.

COBOL/400 Syntax Notation

COBOL/400 basic formats are presented in a uniform system of syntax notation. This notation, designed to assist you in writing COBOL source statements, is explained in the following paragraphs:

- COBOL keywords and optional words appear in uppercase letters; for example:

  ```
  PARM1
  ```

  They must be spelled exactly as shown. If any keyword is missing, the compiler considers it as an error.

- Variables representing user-supplied names or values appear in all lowercase letters; for example:

  ```
  parmx
  ```

- For easier text reference, some words are followed by a hyphen and a digit or a letter, as in:

  ```
  identifier-1
  ```

  This suffix does not change the syntactical definition of the word.

- Arithmetic and logical operators (+, -, *, /, **, >, <, =, > =, and < =) that appear in syntax formats are required. These operators are *special character* reserved words. For a complete listing of reserved COBOL/400 words, see the "Reserved Words" section of the *COBOL/400 Reference* manual.

- All punctuation and other special characters appearing in the diagram are required by the syntax of the format when they are shown; if you leave them out, an error occurs in the program.
- You must write the required clauses and the optional clauses (when used) in the order shown in the diagram unless the associated rules explicitly state otherwise.

How to Read the Syntax Diagrams

Throughout this book, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line:

  The ▶▶— symbol indicates the beginning of a statement.

  The —▶ symbol indicates that the statement syntax is continued on the next line.

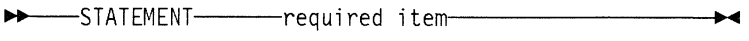
  The ▶— symbol indicates that a statement is continued from the previous line.

  The —▶◀ symbol indicates the end of a statement.

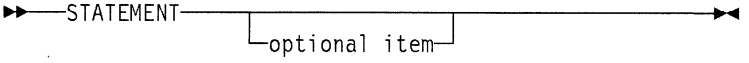
  Diagrams of syntactical units other than statements, such as clauses, phrases and paragraphs, also start with the ▶▶— symbol and end with the —▶◀ symbol.

  Note: Statements within a diagram of an entire paragraph will not start with ▶▶— and end with —▶◀ unless their beginning or ending coincides with that of the paragraph.

- Required items appear on the horizontal line (the main path):

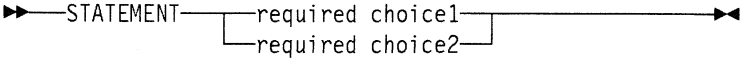


- Optional items appear below the main path:

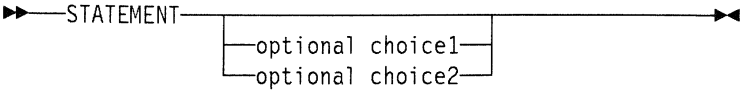


- When you can choose from two or more items, they appear vertically, in a stack.

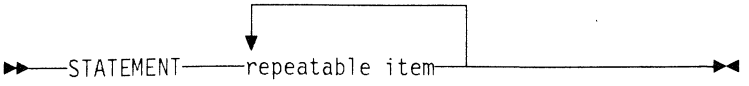
  If you must choose one of the items, one item of the stack appears on the main path:



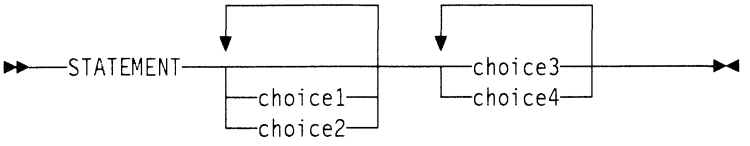
  If choosing one of the items is optional, the entire stack appears below the main path:



- An arrow returning to the left above an item indicates that that item may be repeated:

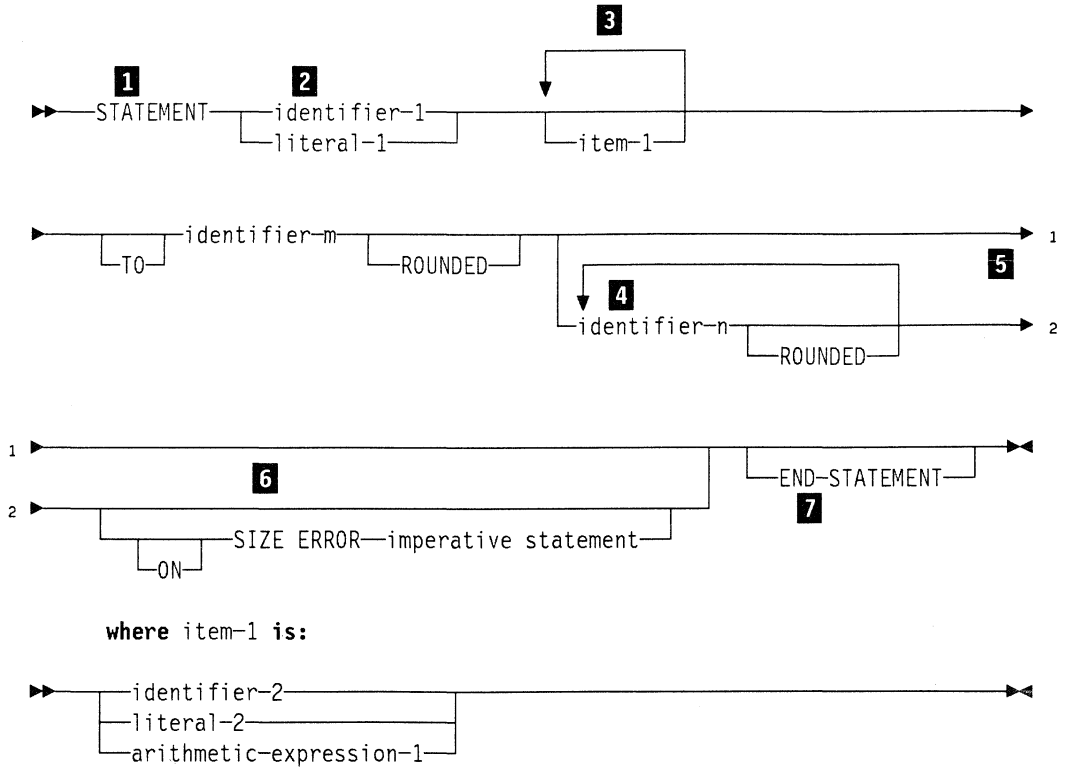


- A repeat arrow above a stack of required or optional choices indicates that you can make more than one choice from the stacked items, or repeat a single choice:



The following example shows how the syntax is used:

Format



1 The STATEMENT keyword must be specified and coded as shown.

2 This operand is required. Either *identifier-1* or *literal-1* must be coded.

3 The operand *item-1* is optional. It may be coded or not, as required by the application. If coded, it may be repeated, with each entry separated by one or more blanks. Entry selections allowed for this operand are described at the bottom of the diagram.

4 The operand *identifier-n* is optional. If specified it may be repeated with one or more blanks separating each entry. Each entry may be assigned the keyword ROUNDED.

5 In cases where multiple lines must be continued past the right margin, line order from top to bottom is preserved.

6 The ON keyword is optional to the keyword SIZE ERROR, which is optional itself. If SIZE ERROR is coded, then the operand *imperative-statement* is required.

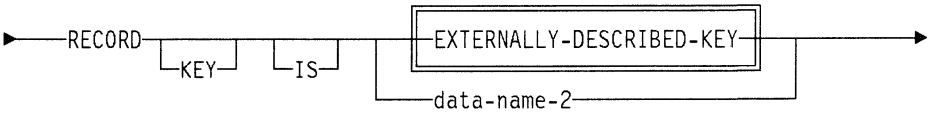
7 The END-STATEMENT keyword may be coded to end the statement. It is not a required delimiter.

IBM Extensions

An IBM extension generally modifies a rule or restriction that immediately precedes it. The standard is presented first, because some programmers use the COBOL/400 language without IBM extensions. The extension is then presented for those who **do** use them.

IBM extensions within figures or tables are shown in boxes unless they are explicitly identified as extensions.

Clauses and statements illustrated within syntax diagrams that are COBOL/400 language extensions to ANSI X3.23-1985 COBOL are enclosed in double lines, as follows:



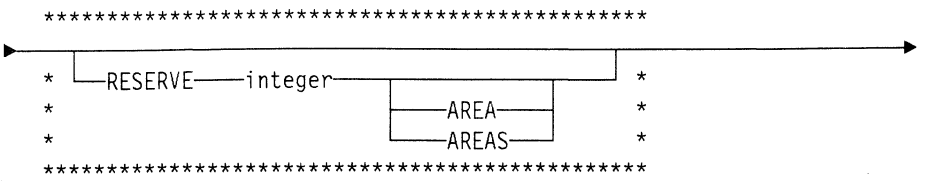
IBM Extension

COBOL/400 language extensions to ANSI X3.23-1985 COBOL that are part of the text description are enclosed in IBM Extension bars, like this paragraph.

End of IBM Extension

Documentary Syntax

COBOL clauses and statements illustrated within syntax diagrams that are syntax checked, but are treated as documentation by the COBOL/400 compiler, are enclosed by asterisks, as follows:



Obsolete Language Elements

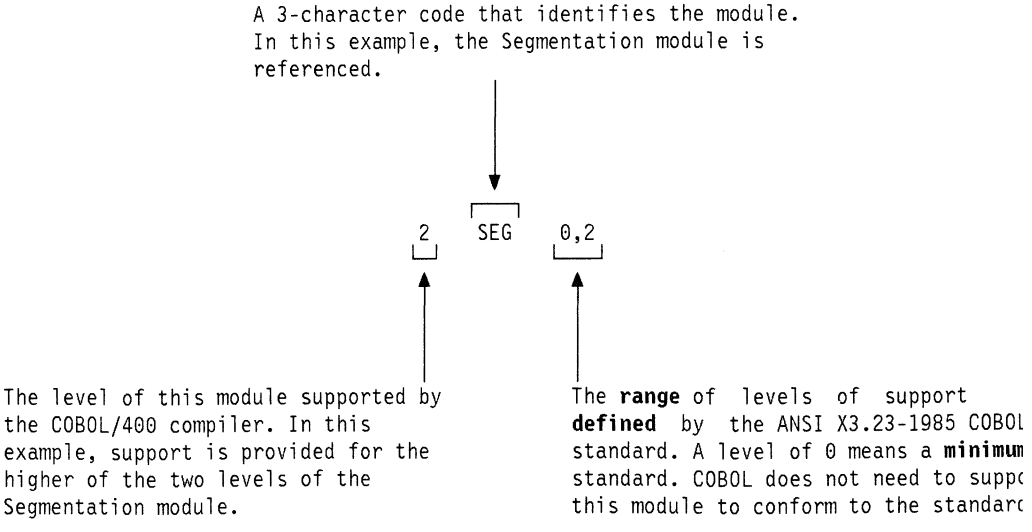
Obsolete language elements are COBOL/400 language elements in the American National Standard Programming Language COBOL, X3.23-1985, that will be deleted from the next revision of this Standard. Throughout this book obsolete language elements are identified in the main description of the language element. They are also listed in the index under 'obsolete elements'.

Industry Standards

The COBOL/400 compiler is designed according to the specifications of the following industry standards as understood and interpreted by IBM as of September 1987.

1. *American National Standard Programming Language COBOL, X3.23-1985*

   Following is an explanation of the notation used in the list:



   The COBOL/400 language conforms to the following modules:

   - Nucleus (1 NUC 1,2)

     Provides internal processing of data within the four basic divisions of a program and the capability for defining and accessing tables.

- Sequential I-O (1 SEQ 1,2)

  Provides access to records of a file in established sequence. The sequence is established as a result of writing the records to the file.

- Relative I-O (1 REL 0,2)

  Provides access to records in either a random or sequential manner; each record is uniquely identified by an integer specifying the record's logical position in a file.

- Indexed I-O (1 INX 0,2)

  Provides access to records in either a random or sequential manner; each record in an indexed file is uniquely identified by the value of a key within that record.

- Inter-Program Communication (1 IPC 1,2)

  Allows a COBOL/400 program to communicate with other programs through transfers of control and access to common data items.

- Sort-Merge (1 SRT 0,1)

  Orders one or more files of records, or combines two or more identically ordered files of records, according to a set of user-specified keys.

- Source-Text Manipulation (1 STM 0,2)

  Allows the insertion of source program text as part of the compilation of the source program. COBOL/400 libraries contain texts which are available to the compiler at compile time and which can be treated by the compiler as part of the source program.

- Debug (2 DEB 0,2)

  Specifies the conditions under which procedures are to be monitored during execution.

- Segmentation (2 SEG 0,2).

  Provides programmer-controlled storage optimization of the Procedure Division by allowing that division to be subdivided both physically and logically.

The following modules are not included:

- Report Writer (0 RPW 0,1)
- Communication (0 COM 0,2).

Portions of this manual are copied (with permission) from the ANSI Standard. Copies of the ANSI Standard may be purchased from:

American National Standards Institute
1430 Broadway
New York, New York 10018

2. *ISO 1989-1985 Standard of the International Organization for Standardization* (ISO)
3. *Federal Standard COBOL of March 1986 (FIPS PUB 21-2), Intermediate Level*
4. The 7-bit coded character sets defined in *American National Standard X3.4-1977, Code for Information Interchange*
5. International Reference Version of the ISO 7-bit code defined in *International Standard 646, 7-Bit Coded Character Set for Information Processing Interchange.*

The COBOL language is developed by the Conference on Data Systems Languages (CODASYL).

Acknowledgment

The following extract from U.S. Government Printing Office Form Number 1965-0795689 is presented for your information and guidance:

> Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention COBOL in acknowledgment of the source, but need not quote this entire section.
>
> COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.
>
> No warranty, expressed or implied, is made by any contributor or by the COBOL 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.
>
> Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.
>
> The authors and copyright holders of copyrighted material:
>
> > FLOW-MATIC** Programming for the UNIVAC (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator, Form No. F28-8013, copyrighted 1959 by IBM; FACT, 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.

Part 1. COBOL Language Structure

Characters

In COBOL, the indivisible unit of data is the character. In the COBOL/400 language, the letters of the alphabet, digits, and special characters that form the COBOL character set are represented by a subset of the EBCDIC (extended binary-coded decimal interchange code) character set.

Figure 1 on page 17 shows the complete set of COBOL characters, their meanings, and their uses.

In COBOL, an **alphabetic character** is a letter or a space character.

In the COBOL/400 language, each lowercase letter is generally equivalent to the corresponding uppercase letter. Important exceptions are letters in nonnumeric literals that do not use hexadecimal notation, and the uppercase D that must appear in column 7 of a debugging line.

The COBOL/400 language is restricted to the defined character set, but the contents of nonnumeric literals, comment lines, comment entries, and the values held in data items, can include any of the characters from the EBCDIC character set.

IBM Extension

Characters from the Double-Byte Character Set (DBCS) are valid characters in certain COBOL character-strings. (See the DBCS information under "Character-Strings" on page 18 for more information.)

End of IBM Extension

Individual characters are joined to form character-strings and separators.

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

A separator is a contiguous string of one or more punctuation characters. A separator can be placed next to another separator, or next to a character-string. Separators are described in detail under "Separators" on page 28.

IBM Extension

You can use the apostrophe in place of the quotation mark if you override the default compiler option.

End of IBM Extension

Character	Meaning	Use
A–Z	Alphabet (uppercase)	Alphabetic characters
a–z	Alphabet (lowercase)	Alphabetic characters
0–9	Arabic numerals	Numeric characters
	Space	Punctuation character
.	Decimal point or Period	Editing character Punctuation character
<	Less than	Relation character
(	Left parenthesis	Punctuation character
+	Plus sign	Arithmetic operator Editing character
$	Dollar sign	Editing character
*	Asterisk	Arithmetic operator Editing character Comment character
)	Right parenthesis	Punctuation character
;	Semicolon	Punctuation character
:	Colon	Punctuation character
–	Minus sign or Hyphen	Arithmetic operator Editing character Continuation character
/	Stroke or Slash	Arithmetic operator Editing character Line control character
,	Comma	Punctuation character Editing character
>	Greater than	Relation character
=	Equal sign	Relation character Punctuation character
"	Quotation marks	Punctuation characters
'	Apostrophe	Punctuation character

Figure 1. COBOL Characters—Their Meanings and Uses

Character-Strings

You can use Single-Byte Character Set (SBCS) character-strings to form:

- COBOL words
- Literals
- PICTURE character-strings
- Comments.

IBM Extension

You can use Double-Byte Character Set (DBCS) character-strings to form:

- Literals
- Comments.

DBCS character-strings are constructed using characters from the Double-Byte Character Set of a system. DBCS character-strings can be embedded into non-numeric strings.

End of IBM Extension

COBOL Words

COBOL words must be character-strings from the set of letters, digits, and the hyphen. The three types of COBOL words are:

- User-defined words
- System-names
- Reserved words.

The maximum length of a COBOL word is 30 characters.

User-Defined Words

A reserved word cannot be a user-defined word. The types of **user-defined words** are listed below, with the rules that must be followed in forming them.

Types of User-Defined Words	General Rules
Alphabet-name Class-name Condition-name Data-name Record-name File-name Index-name Mnemonic-name Pointer-name Routine-name	Each word must contain at least one alphabetic character.
Library-name Program-name Text-name	Each word must contain at least one alphabetic character. The first 10 characters must be unique.
Paragraph-name Section-name	The word need **not** contain an alphabetic character.
Level-numbers: 01-49,66,77,88	Each word must be a 1-digit or 2-digit integer.
Segment-Numbers: 00-99	Must be a 1 or 2 digit integer; it does not have to be unique.

The function of each user-defined word is described in the clause or statement in which it appears.

Valid characters in a user-defined word are:

- A through Z
- a through z
- 0 through 9
- - (hyphen).

The hyphen may not appear as the first or last character in a user-defined word.

User-defined words form sets equivalent to the types in the above table, except that level-numbers are not a set, and condition-names, data-names, and record-names are grouped into a single set. No user-defined word can belong to more than one of these sets. Within each set, each user-defined word must be unique, except as specified in "Methods of Data Reference" on page 39.

System-Names

A **system-name** is a character string that is defined by IBM to have a specific meaning to the system. There are three types of system-names:

- Computer-names
- Language-names
- Implementor-names.

There are two types of implementor-names:

- Environment-name
- Assignment-name.

The meaning of each system-name is described with the format in which it appears.

Reserved Words

A **reserved word** is a character-string with a predefined meaning in a COBOL source program. A reserved word must not be used as a user-defined word or as a system-name. Reserved words can be used only as specified in the formats for a COBOL source program.

COBOL/400 reserved words are listed in Appendix E, "COBOL/400 Reserved Word List" on page 497.

There are five types of reserved words:

- Keywords
- Optional words
- Special characters
- Figurative constants
- Special registers.

Keywords

Keywords are reserved words that are required within a given clause, entry, or statement. Within each format, such words are in uppercase and appear on the main path.

Optional Words

Optional words are reserved words that may be included in the format of a clause, entry, or statement in order to improve readability. They have no effect on the execution of the program. When an optional word is omitted, such as in the **KEY IS** phrase, the meaning of the COBOL statement is unchanged. Optional words are shown in formats as uppercase, but appear below the main path.

Special Characters

There are two types of **special character** reserved words:

- **Arithmetic operators**: + - / * **

  See "Arithmetic Expressions" on page 187.

- **Relational operators**: < > = <= >=

  See "Conditional Expressions" on page 189.

Figurative Constants

Figurative constants are reserved words that name and refer to specific constant values. Figurative constants are not allowed as function arguments. The reserved words for figurative constants and their meanings are:

ZERO/ZEROS/ZEROES
Represents the numeric value zero (0), one or more occurrences of the nonnumeric character zero (0), or the Boolean value B"0", depending on context.

SPACE/SPACES
Represents one or more blanks or spaces; treated as a nonnumeric literal.

HIGH-VALUE/HIGH-VALUES
Represents one or more occurrences of the character that has the highest ordinal position in the collating sequence used. For the EBCDIC and ASCII collating sequences, the character is X'FF'; for other collating sequences, the actual character used depends on the collating sequence. HIGH-VALUE is treated as a nonnumeric literal.

LOW-VALUE/LOW-VALUES
Represents one or more occurrences of the character that has the lowest ordinal position in the collating sequence used. For the EBCDIC and ASCII collating sequences, the character is X'00'; for other collating sequences, the actual character used depends on the collating sequence. LOW-VALUE is treated as a nonnumeric literal.

QUOTE/QUOTES
Represents one or more occurrences of the quotation mark character and must be nonnumeric. QUOTE, or QUOTES cannot be used in place of a quotation mark or an apostrophe to enclose a nonnumeric literal.

IBM Extension

When APOST is specified as a compiler option, the figurative constant QUOTE has the EBCDIC value of an apostrophe.

End of IBM Extension

ALL literal
Represents one or more occurrences of the string of characters comprising the literal. The literal must be a nonnumeric literal or a figurative constant other than the ALL literal.

IBM Extension

ALL Boolean literal
The ALL literal can be Boolean.

End of IBM Extension

When a figurative constant other than ALL literal is used, the word ALL is redundant and is used for readability only. The figurative constant ALL literal must not be used with the INSPECT, STOP, or STRING statements.

Note: 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, is

an obsolete element and is to be deleted from the next revision of the ANSI Standard.

IBM Extension

NULL/NULLS

Represents a value used to indicate that a data item defined with the USAGE IS POINTER clause, ADDRESS OF phrase, or ADDRESS OF special register does not contain a valid address. NULL can be used only where explicitly allowed in the syntax format.

In the COBOL/400 language, a value of NULL is undefined.

End of IBM Extension

The singular and plural forms of a figurative constant are equivalent, and may be used interchangeably. For example, if DATA-NAME-1 is a 5-character data item, each of the following statements will fill DATA-NAME-1 with five spaces:

```
MOVE SPACE TO DATA-NAME-1
MOVE SPACES TO DATA-NAME-1
MOVE ALL SPACES TO DATA-NAME-1
```

A figurative constant may be used wherever `'literal'` appears in a format, except where explicitly prohibited. When a numeric literal appears in a format, the figurative constant ZERO may be used.

IBM Extension

The figurative constant ZERO can be used as a Boolean literal.

End of IBM Extension

The length of a figurative constant depends on the context of the program. The following rules apply:

- When a figurative constant is associated with a data item (for example, when it is moved to or compared with another item), the length of the figurative constant character-string is equal to the size of the associated data item.
- When a figurative constant, other than the ALL literal, is not associated with another data item (for example, in a STOP, STRING, or UNSTRING statement), the length of the character-string is one (1) character.

Special Registers

Special registers are reserved words that name storage areas generated by the compiler. Their primary use is to store information produced through one of the specific COBOL features. Each such storage area has a fixed name, and need not be further defined within the program. Each special register is discussed in the section indicated.

IBM Extension

Special Register	Page
ADDRESS OF	101

Special Register	Page
DB-FORMAT-NAME	218

End of IBM Extension

Special Register	Page
DEBUG-ITEM	Refer to the *COBOL/400 User's Guide*.

IBM Extension

LENGTH OF Special Register: The LENGTH OF special register contains the number of bytes used by a data item referenced by an identifier.

The LENGTH OF phrase creates an implicit special register whose contents equal the current length, in bytes, of the data item referenced by the identifier.

The LENGTH OF special register has the implicit definition:

```
USAGE IS BINARY, PICTURE 9(9)
```

You can use it anywhere in the Procedure Division where you can use a numeric data item having the same definition as the implied definition of the LENGTH OF special register.

It can appear in the starting position or length expression of a reference modifier, but you cannot apply it to an operand that is reference modified.

It **cannot** be either of the following:

- A receiving data item
- A subscript.

You can use LENGTH OF in the BY CONTENT phrase of the CALL statement.

For a table element, the LENGTH OF special register contains the length, in bytes, of one occurrence. To refer to a table element in this case, you do not need to use a subscript.

For a variable-length element, the LENGTH OF special register contains the length based on the current contents of the occurs depending on (ODO) variable.

The register returns a value for any identifier whose length can be determined, even if the area referenced by the identifier is currently not available to the program. For example, an identifier that is part of a 01-level record in a File Definition is not available until the corresponding file is open; however, the LENGTH OF such an identifier can be determined before the file is open.

If, for a variable-length item, the contents of the ODO variable are not available, the LENGTH OF special register is undefined. For example, if an ODO variable is defined in the 01-level record of a file that is not open, no LENGTH OF value exists, and an error results.

A separate LENGTH OF special register exists for each identifier referenced with the LENGTH OF phrase.

For example:

```
MOVE LENGTH OF A TO B
DISPLAY LENGTH OF A, A
ADD LENGTH OF A TO B
CALL "PROGX" USING BY REFERENCE A BY CONTENT LENGTH OF A
```

End of IBM Extension

Special Register	Page
LINAGE-COUNTER	123

IBM Extension

Special Register	Page
WHEN-COMPILED	326

End of IBM Extension

Literals

A **literal** is a character-string whose value is specified either by the characters of which it is composed, or by the use of a figurative constant. (See "Figurative Constants" on page 21.) There are three types of literals: **Boolean**, **nonnumeric**, and **numeric**.

IBM Extension

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 the character 0 or 1. The value of a Boolean literal is the character itself, excluding the delimiting separators. All Boolean literals are of the category Boolean.

End of IBM Extension

Nonnumeric Literals

A **nonnumeric literal** is a character-string enclosed in quotation marks ("), and can contain any allowable character from the EBCDIC character set. The maximum length of a nonnumeric literal is 160 characters.

A nonnumeric literal must be enclosed in quotation marks (") (or apostrophes if the APOST option is in effect). These quotation marks (or apostrophes) are excluded from the literal when the program is compiled. An embedded quotation mark must be represented by a pair of quotation marks ("").

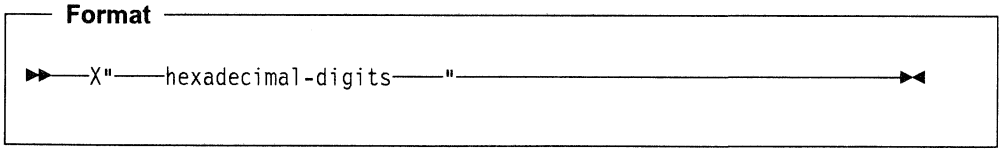
Any punctuation characters included within a nonnumeric literal are part of the value of the literal.

Every **nonnumeric** literal is in the **alphanumeric** data category. (Data categories are described in "Classes and Categories of Data" on page 106.)

IBM Extension

You can use hexadecimal notation to form a **hexadecimal nonnumeric literal**.

Format



X" The opening delimiter for hexadecimal notation of a nonnumeric literal. (If the compiler option *APOST or the PROCESS statement option APOST is specified, the opening delimiter is X'.)

" The closing delimiter for hexadecimal notation of a nonnumeric literal. (If the compiler option *APOST or the PROCESS statement option APOST is specified, the closing delimiter is '.)

Hexadecimal digits can be characters that range from 0 to 9, and A to F, inclusive. Two hexadecimal digits represent a single character, so an even number of hexadecimal digits must be specified in each case.

The maximum length of a hexadecimal nonnumeric literal is 320 hexadecimal digits.

The continuation rules are the same as those for any nonnumeric literal.

The GRAPHIC option of the PROCESS statement does not affect the processing of hexadecimal nonnumeric literals.

The compiler converts the hexadecimal literal into an ordinary nonnumeric literal. Hexadecimal nonnumeric literals can be used anywhere nonnumeric literals can appear.

The padding character for hexadecimal nonnumeric literals is a trailing blank (X"40"). Empty hexadecimal nonnumeric literals are converted to X"40".

End of IBM Extension

IBM Extension

SBCS and DBCS characters can be combined within a character-string to form a **mixed literal**. SAA COBOL statements process mixed literals without sensitivity to the machine representation. Those statements that operate on a byte-to-byte basis (for example, STRING and UNSTRING) may produce literals that are not valid mixtures of SBCS and DBCS. It is the user's responsibility to be certain that the statements are used correctly.

Mixed literals must not be continued.

End of IBM Extension

Numeric Literals

A **numeric literal** is a character-string whose characters are selected from the digits 0 through 9, a sign character (+ or -), and the decimal point. If the literal contains no decimal point, it is an integer. (In this manual, the word **integer** appearing in a format represents a numeric literal of nonzero value that contains no sign and no decimal point; any other restrictions are included with the description of the format.) The following rules apply:

- One through 18 digits are allowed.
- Only one sign character is allowed. If included, it must be the leftmost character of the literal. If the literal is unsigned, it is positive in value.
- Only one decimal point is allowed. If a decimal point is included, it is treated as an **assumed decimal point** (that is, as not taking up a character position in the literal). The decimal point may appear anywhere within the literal except as the rightmost character.
- If enclosed in quotation marks, the compiler treats the literal as a nonnumeric literal.

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

Every **numeric** literal is in the **numeric** data category. (Data categories are described under "Classes and Categories of Data" on page 106.)

PICTURE Character-Strings

A **PICTURE character-string** consists of certain symbols that are composed of the currency symbol and certain combinations of characters in the COBOL character set.

Any punctuation character that 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. (A chart of PICTURE clause symbols appears in Table 10 on page 147.)

Comments

A **comment** is a character-string that can contain any combination of characters from the EBCDIC character set. It has no effect on the execution of the program. There are two forms of comments:

Comment entry
(Identification Division)

This form is described under "Optional Paragraphs" on page 50.

Comment line
(Any division)

This form is described under "Comment Lines" on page 38.

Separators

A separator can be a single punctuation character or a string of punctuation characters.

Table 1 shows the COBOL separator characters supported in the COBOL/400 language.

Table 1. Separators

Separator	Meaning
␣	Space
,␣	Comma
.␣	Period
;␣	Semicolon
(	Left parenthesis
)	Right parenthesis
"	Quotation mark
==	Pseudo-text delimiter
:	Colon
IBM Extension ' End of IBM Extension	IBM Extension Apostrophe End of IBM Extension
IBM Extension B" End of IBM Extension	IBM Extension Opening delimiter for Boolean literal End of IBM Extension
IBM Extension X" End of IBM Extension	IBM Extension Opening delimiter for hexadecimal nonnumeric literal End of IBM Extension

Rules for Separators

In the following description, brackets enclose each separator. Anywhere a space is used as a separator, or as part of a separator, more than one space may be used.

A space [␣]

A space can immediately precede or follow any separator except:

- As specified in standard format rules (see "Reference Format" on page 33).

- Within quotation marks. (or apostrophes if the APOST option is in effect). Spaces between quotation marks are considered part of the nonnumeric literal; they are not considered separators.

Period [.b]
Comma [,b]
Semicolon [;b]

The separator period must be used only to indicate the end of a sentence, or as shown in formats. The separator comma and separator semicolon may be used anywhere the separator space is used.

- In the **Identification Division**, separator commas and separator semicolons can be used in the comment-entries. Each paragraph must end with a separator period.
- In the **Environment Division**, separator commas or separator semicolons may separate clauses and operands within clauses. The SOURCE-COMPUTER, OBJECT-COMPUTER, SPECIAL-NAMES, and I-O-CONTROL paragraphs must each end with a separator period. In the FILE-CONTROL paragraph, each File-Control entry must end with a separator period.
- In the **Data Division**, separator commas or separator semicolons may separate clauses and operands within clauses. File (FD), Sort/Merge file (SD), and data description entries must each end with a separator period.
- In the **Procedure Division**, separator commas or separator semicolons may separate statements within a sentence, and operands within a statement. Each sentence and each procedure must end with a separator period.

Parentheses [( ] . . . [ )]

Must appear as balanced pairs of left and right parentheses. They delimit subscripts, reference modification, arithmetic expressions, and conditions.

Quotation marks [b"] . . . ["b]

An opening quotation mark must be immediately preceded by a space or a left parenthesis. A closing quotation mark must be immediately followed by a separator (space, comma, semicolon, period, or right parenthesis). Quotation marks must appear as balanced pairs. They delimit nonnumeric literals, except when the literal is continued (see "Continuation Lines" on page 37).

IBM Extension

Under the *APOST compiler option, or the APOST PROCESS option, an apostrophe can be used in place of a quotation mark.

End of IBM Extension

Pseudo-text delimiters [b==] ... [==b]
An opening pseudo-text delimiter must be immediately preceded by a space. A closing pseudo-text delimiter must be immediately followed by a separator (space, comma, semicolon, or period). Pseudo-text delimiters must appear as balanced pairs. They delimit pseudo-text. (See "COPY Statement" on page 456.)

Colon [:]
The colon is a separator, and is required when shown in general formats.

IBM Extension

B" is a separator when used to describe a Boolean literal. The B must immediately precede the quotation mark.

End of IBM Extension

IBM Extension

X" is a separator when used to describe a hexadecimal nonnumeric literal. The X must immediately precede the quotation mark.

End of IBM Extension

Note: Any punctuation character included in a PICTURE character-string, a comment character-string, or a nonnumeric literal is not considered as a punctuation character, but rather as part of the character-string or literal.

Sections and Paragraphs

Sections and paragraphs define a program. They are subdivided into clauses and statements. For more information on sections, paragraphs, and statements, see "Procedures" on page 185.

Unless the associated rules explicitly state otherwise, each required clause or statement must be written in the sequence shown in its format. If optional clauses or statements are used, they must be written in the sequence shown in their formats. These rules are true even for clauses and statements treated as comments.

The grammatical hierarchy follows this form:

- Identification Division
  - Paragraphs
    - Entries
      - Clauses
- Environment Division
  - Sections
    - Paragraphs
      - Entries
        - Clauses
          - Phrases
- Data Division
  - Sections
    - Entries
      - Clauses
        - Phrases
- Procedure Division
  - Sections
    - Paragraphs
      - Sentences
        - Statements
          - Phrases

Entries

An **entry** is a series of clauses ending with a separator period. Entries are constructed in the Identification, Environment, and Data Divisions.

Clauses

A **clause** is an ordered set of consecutive COBOL character-strings that specifies an attribute of an entry. Clauses are constructed in the Identification, Environment, and Data Divisions.

Sentences

A **sentence** is a sequence of one or more statements, ending with a separator period. Sentences are constructed in the Procedure Division.

Statements

A **statement** is a valid combination of a COBOL verb and its operands. It specifies an action to be taken by the object program. Statements are constructed in the Procedure Division. For descriptions of the different types of statements, see:

- "Imperative Statements" on page 204
- "Conditional Statements" on page 206
- "Delimited Scope Statements" on page 207
- "Part 4. Compiler-Directing Statements" on page 453.

Phrases

Each clause or statement in the program can be subdivided into smaller units called **phrases**.

Reference Format

COBOL programs **must** be written in the COBOL reference format. Figure 2 shows the reference format for a COBOL source line.

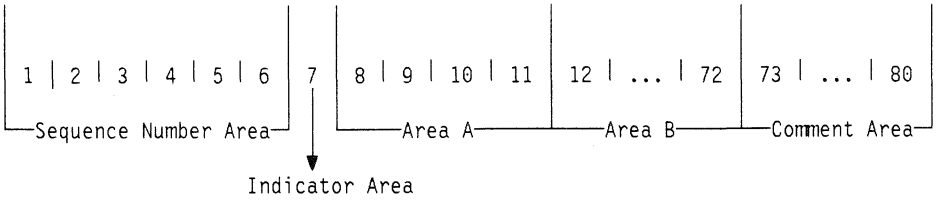


Figure 2. Reference Format for COBOL Source Line

The following areas are described below in terms of an 80-character line:

Sequence Number Area
Columns 1 through 6

Indicator Area
Column 7

Area A
Columns 8 through 11

Area B
Columns 12 through 72

Comment Area
Columns 73 through 80 are available for your own use; for example, to identify your program.

Sequence Number Area (Columns 1 through 6)

A sequence number identifies each statement to be compiled by the COBOL compiler. The use of sequence numbers is optional. If used, a sequence number must be in columns 1 through 6, and may consist of any character in the character set of the computer.

If sequence numbers are present in the source program, they may be in any order, and they need not be unique.

IBM Extension

The user can use sequence checking at compile time by specifying SEQUENCE.

If the NUMBER option is specified, the sequence numbers from columns 1 through 6 are used; otherwise the source sequence numbers provided in the source file are used.

End of IBM Extension

Indicator Area (Column 7)

Use the indicator area to specify:

- The continuation of words or nonnumeric literals from the previous line onto the current line
- The treatment of text as documentation
- Debugging lines.

See "Continuation Lines" on page 37, "Comment Lines" on page 38, and "Debugging Lines" on page 38.

Area A (Columns 8 through 11)

The following items must begin in Area A:

- Division header
- Section header
- Paragraph header or paragraph name
- Level indicator or level-number (01 and 77)
- DECLARATIVES and END DECLARATIVES.

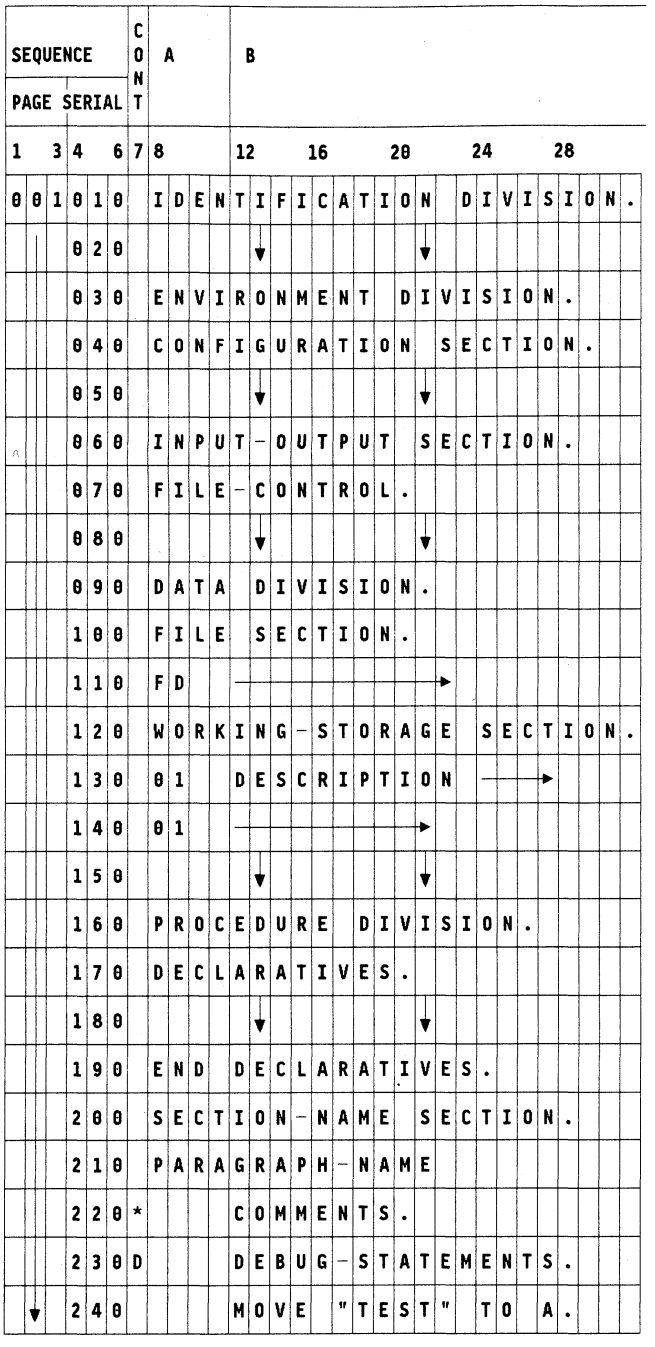


Figure 3. Basic Skeleton of a COBOL Program

Division Header

A division header is a combination of words, followed by a separator period, that indicates the beginning of a division:

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

PROCEDURE DIVISION.

A division header (except when a USING phrase is specified with a Procedure Division header) must be immediately followed by a separator period. Except for the USING phrase, no text may appear on the same line.

Section Header

In the Environment and Procedure Divisions, a section header indicates the beginning of a series of paragraphs; for example:

```
FILE-CONTROL.
DECLARATIVES.
```

In the Data Division, a section header indicates the beginning of an entry; for example:

```
FILE SECTION.
```

A section header must be immediately followed by a period except when Procedure Division segment numbers are specified.

Paragraph Header or Paragraph Name

A paragraph header or paragraph name indicates the beginning of a paragraph.

In the Environment Division, a paragraph consists of a paragraph header followed by one or more entries. For example:

```
OBJECT-COMPUTER. computer-name
```

In the Procedure Division, a paragraph consists of a paragraph-name followed by one or more sentences.

Level Indicator (FD and SD) or Level-Number (01 and 77)

A level indicator can be either FD or SD. It must begin in Area A and be followed by a space. (See "File Section" on page 115.)

A level-number that must begin in Area A is a 1- or 2-digit integer with a value of 01 or 77. For more information, see "Level-Numbers" on page 128.

DECLARATIVES and END DECLARATIVES

DECLARATIVES and END DECLARATIVES are key words that begin and end the declaratives part of the source program.

In the Procedure Division, each of the key words DECLARATIVES and END DECLARATIVES must begin in Area A and be followed immediately by a separator period; no other text may appear on the same line. After the key words END DECLARATIVES, no text may appear before the following section header. (See "Declaratives" on page 184.)

Area B (Columns 12 through 72)

The following items must begin in Area B:

- Entries, sentences, statements, clauses
- Continuation lines.

Entries, Sentences, Statements, Clauses

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

Within an entry or sentence, successive lines in Area B may have the same format, or may be indented to clarify program logic. The output listing is indented only if the input statements are indented. Indentation does not affect the meaning of the program. The programmer can choose the amount of indentation, subject only to the restrictions on the width of Area B. See also "Sections and Paragraphs" on page 31.

Continuation Lines

Any sentence, entry, clause, or phrase that requires more than one line can be continued in Area B of the next line that is neither a comment line nor a blank line. The line being continued is a **continued line**; the succeeding lines are **continuation lines**. Area A of a continuation line must be blank.

IBM Extension

Mixed literals may not be continued.

End of IBM Extension

If there is no hyphen (-) in the indicator area (column 7) of a line, the last character of the preceding line is assumed to be followed by a space.

If there is a hyphen in the indicator area of a line, the first nonblank character of this continuation line immediately follows the last nonblank character of the continued line without an intervening space.

If the continued line contains a nonnumeric literal without a closing quotation mark, all spaces at the end of the continued line (through column 72) are considered to be part of the literal. The continuation line must contain a hyphen in the indicator area, and the first nonblank character must be a quotation mark. The continuation of the literal begins with the character immediately following the quotation mark.

If the last character on the continued line of a nonnumeric literal is a single quotation mark in column 72, the continuation line must start with two consecutive quotation marks. This will result in a single quotation mark as part of the value of the nonnumeric literal.

For the pseudo-text delimiter separator (==), the two characters that make up the separator must occupy the same line.

Area A or Area B

The following items may begin in either Area A or Area B:

- Level-numbers
- Comment lines
- Debugging lines
- Blank lines
- Pseudo-text.

Level-Numbers

A level-number that may begin in Area A or B is a 1- or 2-digit integer with a value of 02 through 49; 66, or 88. For more information, see "Level-Numbers" on page 128.

Comment Lines

A **comment line** is any line with an asterisk (*) or slash (/) in the indicator area (column 7) of the line. The comment may be written anywhere in Area A and Area B of that line, and may consist of any combination of characters from the EBCDIC character set. A comment line may be placed anywhere in the program following the Identification Division header.

Multiple comment lines are allowed. Each must begin with either an asterisk (*) or a slash (/) in the indicator area.

An asterisk (*) comment line is printed in the output listing, immediately following the last preceding line. A slash (/) comment line is printed on the first line of the next page, and the current page of the output listing is ejected.

The compiler treats a comment line as documentation, and does not check it syntactically.

Debugging Lines

A **debugging line** is any line with a 'D' in the indicator area of the line. Debugging lines can be written in the Environment Division (after the OBJECT-COMPUTER paragraph), the Data Division, and the Procedure Division. If a debugging line contains only spaces in Area A and Area B, it is considered a blank line.

See "WITH DEBUGGING MODE" on page 54.

Blank Lines

A **blank line** contains nothing but spaces from column 7 through column 72. A blank line may appear anywhere in a program.

Pseudo-Text

The character-strings and separators comprising **pseudo-text** may start in either Area A or Area B. If, however, there is a hyphen in the indicator area (column 7) of a line which follows the opening pseudo-text delimiter, Area A of the line must be blank, and the rules for continuation lines apply to the formation of text words.

Methods of Data Reference

References to data and procedures can be either explicit or implicit. This section contains the rules for qualification and for explicit and implicit data references.

Every user-defined name in a COBOL program is assigned by the user to name a resource for solving a data processing problem. To use a resource, a statement in a COBOL program must contain a reference that uniquely identifies that resource. To ensure uniqueness of reference, a user-defined name can be qualified, subscripted, or reference-modified.

In the syntax diagrams, the term **identifier** refers to a syntactically correct combination of a data-name, with its qualifiers, subscripts, and reference modifiers as required for uniqueness of reference, that names a data item. Rules for identifiers associated with a format may, however, specifically prohibit qualification, subscripting, or reference modification. The term **data-name** refers to a name that must not be qualified, subscripted, or reference modified, unless specifically permitted by the rules for the format.

Qualification

A name can be made unique if it exists within a hierarchy of names, and the name can be identified by specifying one or more higher-level names in the hierarchy. The higher-level names are called **qualifiers**, and the process by which such names are made unique is called **qualification**.

Qualification is specified by placing one or more phrases after a user-specified name, with each phrase made up of the word IN or OF followed by a qualifier. (IN and OF are logically equivalent.)

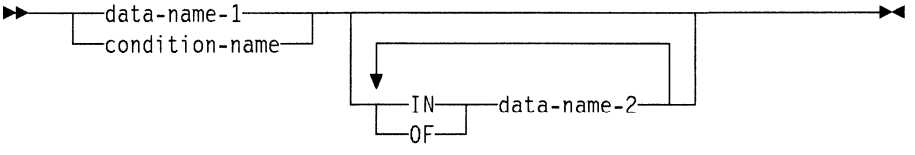
In any hierarchy, the data name associated with the highest level must be unique, and cannot be qualified.

The programmer must specify enough qualification to make the name unique; however, it may not be necessary to specify all the levels of the hierarchy. For example, if there is more than one file whose records contain the field EMPLOYEE-NO, but only one of the files has a record named MASTER-RECORD:

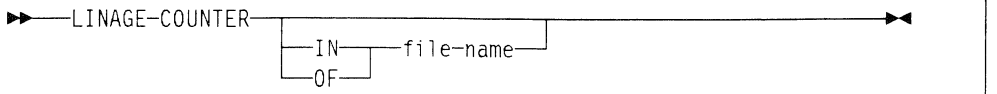
- EMPLOYEE-NO OF MASTER-RECORD sufficiently qualifies EMPLOYEE-NO
- EMPLOYEE-NO OF MASTER-RECORD OF MASTER-FILE is valid but unnecessary.

References to Data Division Names

Format 1



Format 2

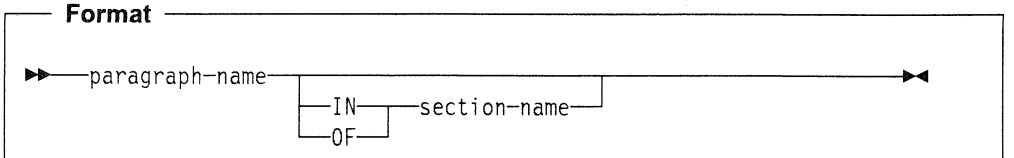


Data Division names that are explicitly referenced in a program must be either uniquely defined, or made unique through qualification. Unreferenced data-names need not be uniquely defined.

A data-name associated with a level-number 01, or with an FD or SD level indicator in the File Section, is the highest level in a data hierarchy. If referenced, it must be uniquely defined, because it cannot be qualified. Data items with level-numbers 02 through 49 are successively lower levels in a data hierarchy, and if referenced, must be either uniquely defined, or made unique through qualification. Level-77 data-names, if referenced, must be uniquely defined, because they cannot be qualified.

References to Procedure Division Names

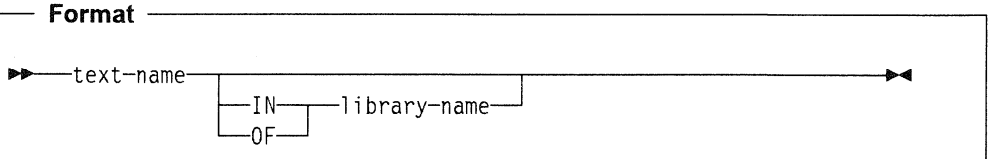
Format



Procedure Division names that are explicitly referenced in a program must be unique within a section. A section-name, described on page 185, is the highest (and only) qualifier available for a paragraph-name and must be unique.

References to COPY Libraries

Format



For rules on referencing COPY libraries, see "COPY Statement" on page 456.

Qualification Rules

The rules for qualifying a name are:

- A name **can** be qualified even though it does not **need** qualification.
- Each qualifier must be of a higher level than the name it qualifies, and must be within the same hierarchy.

  For example:

```
01 FIELD-A
   02 FIELD-B
      05 SUB1
         07 SUB2
   02 FIELD-C
         07 SUB1
```

  A hierarchy includes all subordinate entries to the next equal or higher level-number. Therefore, in the above example all entries are in the hierarchy of FIELD-A. All entries from FIELD-B to, but not including, FIELD-C are in the hierarchy of FIELD-B.

  In the hierarchy of FIELD-A, SUB1 can be used twice; once as subordinate to FIELD-B and once as subordinate to FIELD-C. In references to SUB-1, it must be qualified as SUB-1 OF FIELD-B or SUB-1 OF FIELD-C. Within FIELD-B or FIELD-C, SUB1 cannot be subordinate to itself.

- The complete list of qualifiers for one data-name must not be the same as a partial list of qualifiers for another.
- If a data-name or a condition-name is assigned to more than one data item, it must be qualified each time it is referred to (for the one exception, see "REDEFINES Clause" on page 158).
- A data-name cannot be subscripted when it is being used as a qualifier.
- If referenced in the program, data-names can be duplicated only in those places where they can be made unique through qualification.
- If there is more than one combination of qualifiers that ensures uniqueness, then any of these combinations can be used.
- The data-name may be a record-name.
- If referenced in the program, a section-name must not be duplicated.
- If referenced in the program, 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 within the section in which it appears.
- 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.
- Library-name must be unique in the system. Therefore, the first 10 characters of library-name must be unique.
- Text-name (member-name) must be qualified by the library-name and file-name in which it resides. If no library is specified, the library list is searched.
- A maximum of 48 qualifiers (49 qualifiers for file data) can be specified.

IBM Extension

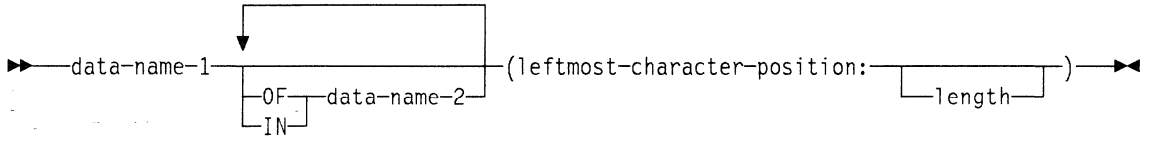
File-name is optional for the COPY statement, Format 1. If file-name is not specified, the default is QLBLSRC.

End of IBM Extension

Reference Modification

Reference modification resembles substringing in other computer languages. Reference modification defines a data item by specifying a starting position and length for the item.

Format



data-name-1
Must refer to a non-Boolean data item whose implicit or explicit usage is DISPLAY. Data-name-1 can be qualified or subscripted.

leftmost-character-position
Must be a numeric literal or an arithmetic expression. The evaluation of the leftmost-character-position must result in a positive nonzero integer that is less than or equal to the number of characters in the data item referenced by data-name-1.

length
Must be a numeric literal or an arithmetic expression.

The evaluation of length must result in an integer that is greater than zero, and less than 32 768. If length is not specified, the unique data item created 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.

Note: If the result of an arithmetic expression is something other than an integer, truncation occurs, resulting in an integer.

Reference modification is generally allowed anywhere an identifier referencing an alphanumeric data item is allowed.

Each character of a data item referenced by data-name-1 is assigned an ordinal number incrementing by one from the leftmost position to the rightmost position. The leftmost position is assigned the ordinal number of 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.

If the data item referenced by data-name-1 is described as numeric, numeric-edited, 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.

Without reference modification, you cannot use operands that are longer than 32 767 bytes in the following statements:

ACCEPT
CANCEL
DISPLAY
EVALUATE (relational condition)
IF (relational condition)
PERFORM (relational condition)
STRING
UNSTRING
USE (USE FOR DEBUGGING)

Note: When you use `IF condition-name`, an implicit comparison of the condition-name (88-level item) with its condition variable takes place, even though the condition variable is not referenced in the IF statement. Therefore, `IF condition-name` cannot be used when the associated condition variable is longer than 32 767 bytes.

With reference modification, this 32 767-byte limit does not apply if the length of the operand is known at compilation time. In other words, the reference modification length specification (or starting position in the absence of a length specification) must be a numeric literal.

Evaluation of Operands

Reference modification for an operand is evaluated as follows:

- If subscripting is specified for the operand, the reference modification is evaluated immediately after evaluation of the subscript.
- If subscripting is not specified for the operand, the reference modification is evaluated at the time subscripting would be evaluated if subscripts had been specified.

Reference modification creates a unique data item which is a subset of the data item referenced by data-name-1. This unique data item is considered an elementary data item without the JUSTIFIED clause. It 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 class and category alphanumeric.

For example:

```
MOVE whole-name(1:25) TO last-name
```

This example transfers the first 25 characters in the variable `whole-name` to the variable `last-name`.

Range Errors

An out-of-range reference modification component, such as a leftmost-character-position of zero, causes system message MCH0603. This is the same message that signals errors in subscript ranges and character-string boundaries.

Restrictions on Reference Modification

The INDICATORS phrase does not support reference modification, and the same is generally true of record-names.

The following statements do not support reference modification:

- INSPECT
- Extended ACCEPT
- Extended DISPLAY

Table 2 shows some additional restrictions.

Table 2. Additional Restrictions on Reference Modification

Statement	Restriction
STRING	You cannot reference modify identifier-3.
UNSTRING	You cannot reference modify identifier-1.
USE FOR DEBUGGING	You cannot reference modify identifier-1.
START	You can reference modify last occurrence of data-name-1 only.

Subscripting

Subscripts can be used only when reference is made to an element within a table of elements that have not been assigned individual data-names. See "Subscripting" on page 143 and "Subscripting Using Index-Names (Indexing)" on page 144.

Data Attribute Specification

Explicit data attributes are those you specify in actual COBOL coding.

Implicit data attributes are default values. If you do not explicitly code a data attribute, the compiler assumes a default value.

For example, you need not specify the USAGE of a data item. If it is omitted, the default is USAGE DISPLAY, which is the implicit data attribute. If, however, you specify USAGE DISPLAY in COBOL coding, it becomes an explicit data attribute.

Transfer of Control

In the Procedure Division, unless there is an **explicit** control transfer or there is no next executable statement, program flow transfers control from statement to statement in the order in which the statements are written. (See Note below.) This normal program flow is an **implicit** transfer of control.

In addition to the implicit transfers 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. The following examples show **implicit** transfers of control, overriding statement-to-statement transfer of control:

- After execution of the last statement of a procedure being executed under control of another COBOL statement, control implicitly transfers. (COBOL statements that control procedure execution are, for example: MERGE, PERFORM, SORT, and USE.)
- During SORT or MERGE statement execution, when control is implicitly transferred to an input or output procedure.
- During execution of any COBOL statement that causes execution of a declarative procedure, control is implicitly transferred to that procedure.
- At the end of execution of any declarative procedure, control is implicitly transferred back to the control mechanism associated with the statement that caused its execution.

COBOL also provides **explicit** control transfers through the execution of any procedure branching or conditional statement.

Note: The term "next executable statement" refers to the next COBOL statement to which control is transferred, according to the rules given above. There is no **next executable statement** under these circumstances:

- When the program contains no Procedure Division.
- Following 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.
- Following 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.
- Following 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.
- Following a STOP RUN, EXIT PROGRAM, or GOBACK statement that transfers control outside the COBOL program.

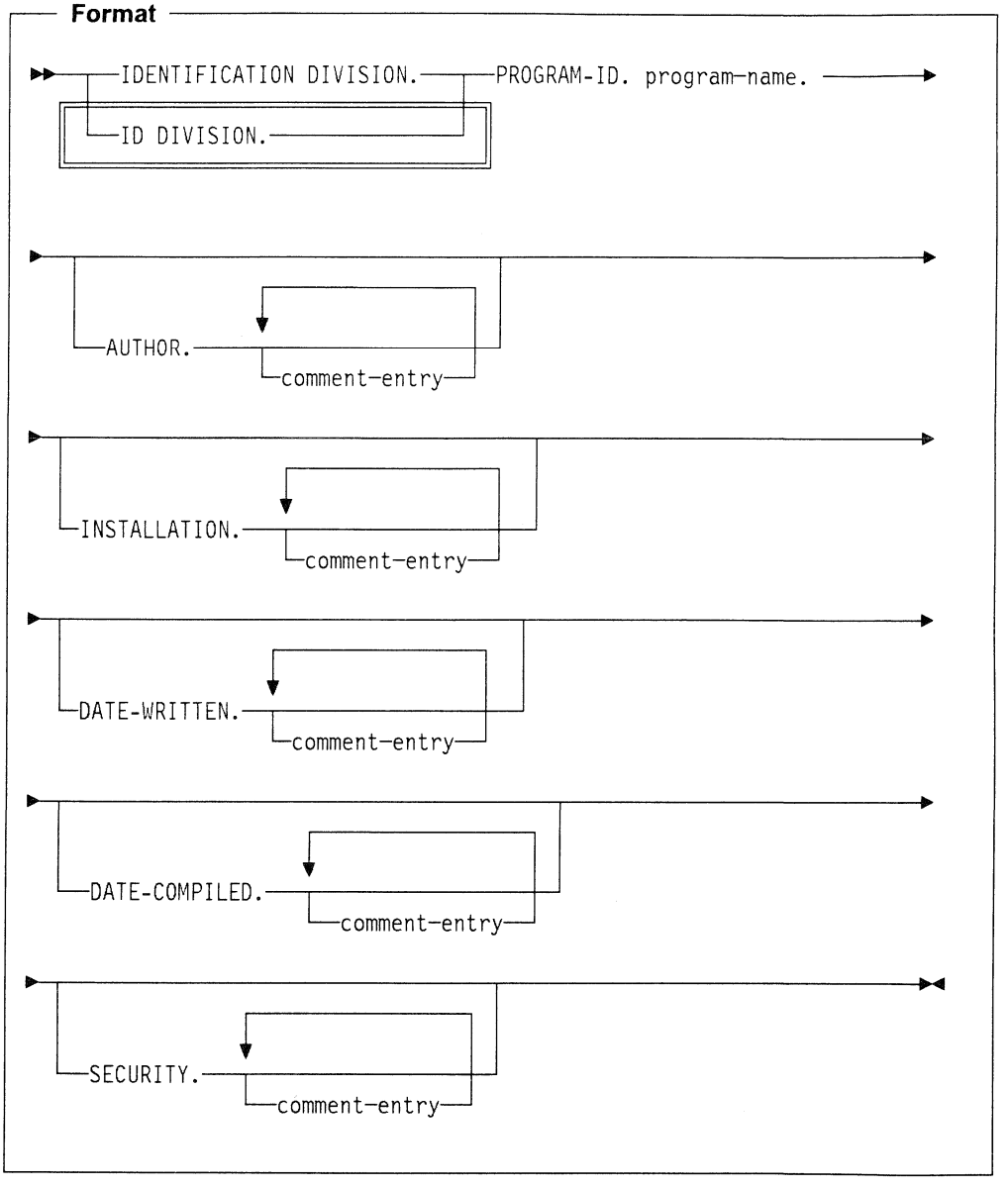
When there is no next executable statement and control is not transferred outside the COBOL program, the program flow of control is undefined unless the program execution is in the nondeclarative procedures portion of a program under control of a CALL statement, in which case an implicit EXIT PROGRAM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

Part 2. COBOL Program Structure

Identification Division

The Identification Division must be the first division in every COBOL source program. It names the program and may include the date the program was written, the date of compilation, and other such documentary information about the program.

Format



The Identification Division must begin with the words IDENTIFICATION DIVISION followed by a separator period.

The first paragraph of the Identification Division must be the PROGRAM-ID paragraph. The other paragraphs are optional, but, when written, must appear in the order shown in the format.

IBM Extension

The abbreviation ID DIVISION may be substituted for the standard division header, and the optional paragraphs may be in any order.

End of IBM Extension

Note: The SEU Syntax Checker requires that the first sentence of the following paragraph headers begin on the same line as the paragraph header:

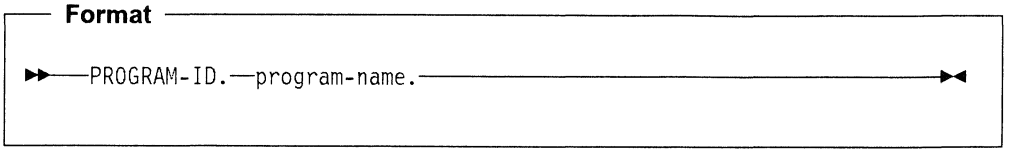
- PROGRAM-ID
- AUTHOR
- INSTALLATION
- DATE-WRITTEN
- DATE-COMPILED
- SECURITY

Coding Example

```
IDENTIFICATION DIVISION.
PROGRAM-ID. IDSAMPLE.
AUTHOR. PROGRAMMER NAME.
INSTALLATION. COBOL DEVELOPMENT CENTER.
DATE-WRITTEN. 08/27/88.
DATE-COMPILED. 09/01/88 12:57:53.
SECURITY. NON-CONFIDENTIAL.
```

PROGRAM-ID Paragraph

Format



The PROGRAM-ID paragraph specifies the name by which the object program is known to the system. It is required and must be the first paragraph in the Identification Division.

program-name
A user-defined word that identifies your object program to the system. The system uses the first 10 characters of program-name as the identifying name of the program; these first 10 characters, therefore, should be unique among program-names. The first character of program-name is forced to be alphabetic; if it is numeric, it is converted as follows:

0	to	J
1 through 9	to	A through I

If a hyphen is in positions 2 through 10, it is converted to zero (0).

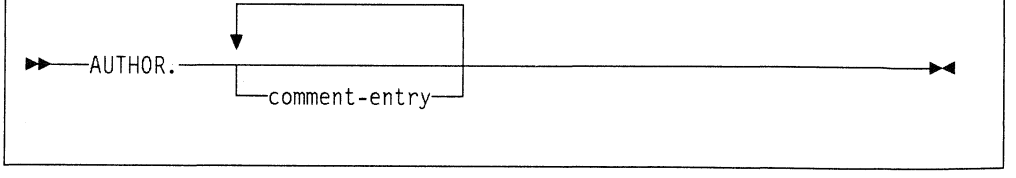
The name by which the program is known to the system can be overridden by the PGM parameter of the CRTCBLPGM command. See the *COBOL/400 User's Guide* for more information on the PGM parameter.

Optional Paragraphs

These optional paragraphs in the Identification Division may be omitted:

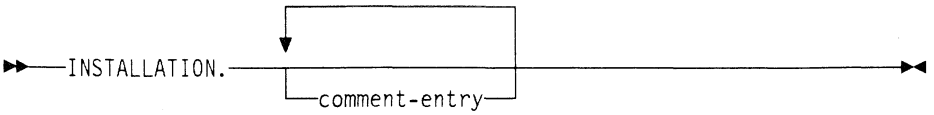
AUTHOR
Name of the author of the program.

Format



INSTALLATION
Name of the company or location.

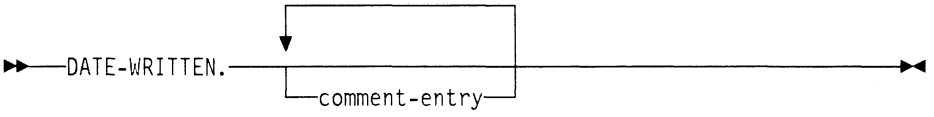
Format



DATE-WRITTEN

Date the program was written.

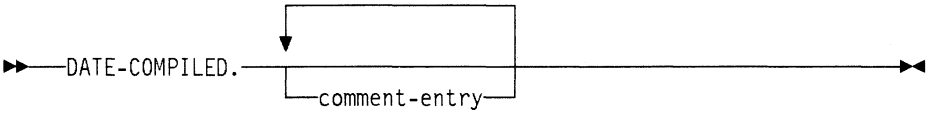
Format



DATE-COMPILED

Date the program was compiled.

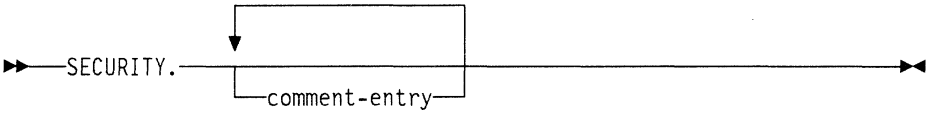
Format



SECURITY

Level of confidentiality of the program.

Format



Note: These optional paragraphs are obsolete elements and are to be deleted from the next revision of the ANSI Standard.

The **comment-entry** in any of the optional paragraphs may be any combination of characters from the character set of the computer. The comment-entry is written in Area B on one or more lines.

The paragraph name DATE-COMPILED and any comment-entry associated with it are replaced during compilation with a paragraph of the form:

```
DATE-COMPILED. current date.
```

Comment-entries serve only as documentation; they do not affect the meaning of the program. A hyphen in the indicator area (column 7) is not permitted in comment-entries.

IBM Extension

Comment-entry may contain the *CBL, *CONTROL, EJECT, SKIP1, SKIP2, SKIP3, or TITLE statements anywhere on the line. These statements will be acted on if they are alone on a line within the comment-entry, and they will not terminate the comment-entry.

End of IBM Extension

IBM Extension

Comments may combine DBCS and SBCS character-strings. Multiple lines are allowed in a comment-entry containing DBCS strings.

Note: DBCS strings are described under "Character-Strings" on page 18.

End of IBM Extension

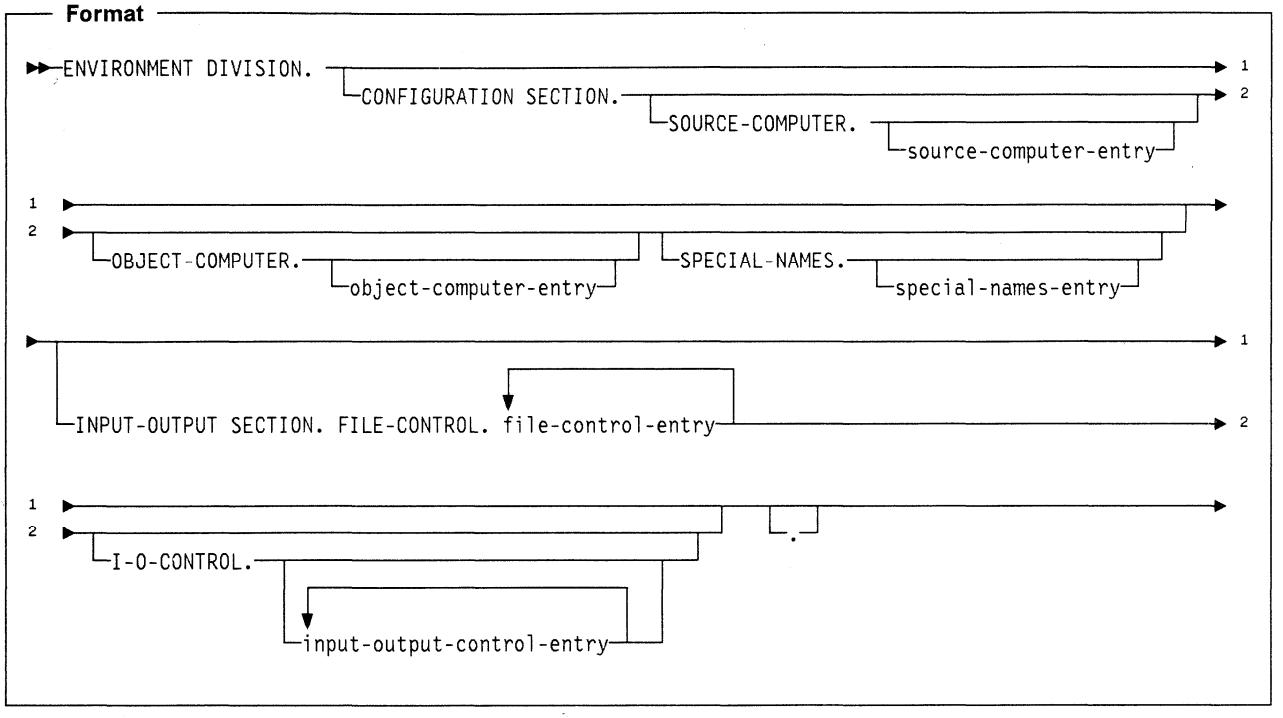
Environment Division—Configuration Section

The Environment Division has two sections:

- The Configuration Section
- The Input-Output Section. (See "Environment Division—Input-Output Section" on page 70.)

The Environment Division is optional in a COBOL source program.

Format



The Configuration Section is optional. When specified, it can describe the computer on which the source program is compiled and the computer on which the object program is executed.

In addition, the Configuration Section can:

- Relate IBM-defined environment-names to user-defined mnemonic names
- Specify the collating sequence
- Specify a substitution for the currency sign
- Interchange the functions of the comma and the period in PICTURE clauses and numeric literals
- Relate alphabet-names to character sets or collating sequences
- Relate class names to sets of characters.

Each paragraph must contain one, and only one, separator period immediately after the last entry in the paragraph.

Note: The SEU Syntax Checker requires that the first clause of the following paragraphs be entered on the same line as the paragraph name:

- SOURCE-COMPUTER

- OBJECT-COMPUTER
- SPECIAL-NAMES

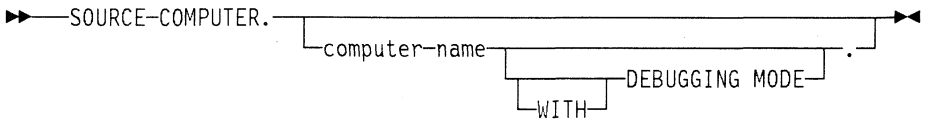
Coding Example

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-AS400.
OBJECT-COMPUTER. IBM-AS400.
SPECIAL-NAMES. C01 IS TOP-OF-PAGE.
```

SOURCE-COMPUTER Paragraph

The SOURCE-COMPUTER paragraph describes the computer on which the source program is to be compiled.

Format



computer-name
A system-name. The suggested computer-name is:

```
IBM-AS400.
```

WITH DEBUGGING MODE
Activates a compile-time switch for debugging lines written in the source program.

A debugging line is a statement that is compiled only when the compile-time switch is activated. Debugging lines allow you, for example, to check the value of a data-name at certain points in a procedure.

The WITH DEBUGGING MODE clause causes any USE FOR DEBUGGING procedure to be compiled. Without this clause, these procedures are treated as comments and ignored.

Note that the USE FOR DEBUGGING declarative causes all subsequent statements to be ignored up to a valid USE AFTER EXCEPTION/ERROR statement, or END DECLARATIVES delimiter. Entire programs can be ignored because of this.

To specify a debugging line in your program, code a 'D' in column 7 (indicator area). You may include successive debugging lines, but each must have a 'D' in column 7 and you may not break character strings across lines.

All your debugging lines must be written so that the program is syntactically correct, whether the debugging lines are compiled or treated as comments.

The presence or absence of the DEBUGGING MODE clause is logically determined after all COPY statements are processed.

You may code debugging lines in the Environment (after the OBJECT-COMPUTER paragraph), Data, or Procedure Divisions.

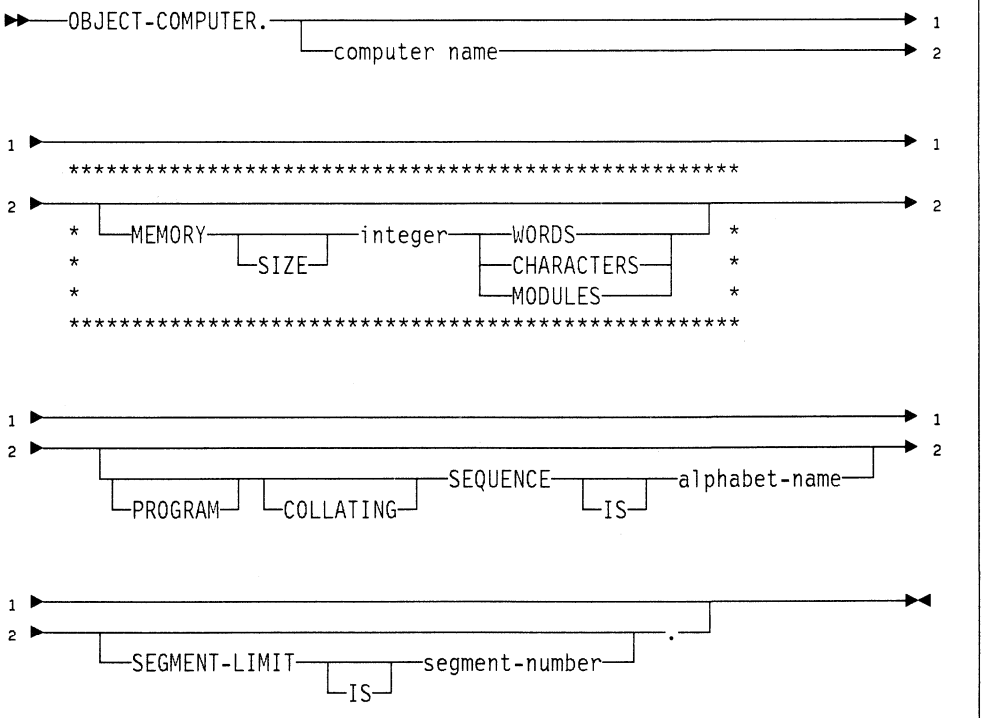
If a debugging line contains only spaces in Area A and in Area B, it is treated the same as a blank line.

Except for the WITH DEBUGGING MODE clause, the SOURCE-COMPUTER paragraph is syntax-checked, but has no effect on the execution of the program.

OBJECT-COMPUTER Paragraph

The OBJECT-COMPUTER paragraph specifies the system for which the object program is designated.

Format



computer-name

A system-name. The suggested computer-name is:

```
IBM-AS400.
```

MEMORY SIZE

The amount of main storage needed to run the object program. The MEMORY SIZE clause is syntax-checked, but it has no effect on the execution of the program.

Note: MEMORY SIZE is an obsolete element, and is to be deleted from the next revision of the ANSI Standard.

integer
Expressed in words, characters, or modules.

PROGRAM COLLATING SEQUENCE IS
The collating sequence used in this program is the collating sequence associated with the specified alphabet-name.

alphabet-name
The collating sequence.

PROGRAM COLLATING SEQUENCE determines the truth value of the following nonnumeric comparisons:

- Those explicitly specified in relation conditions
- Those explicitly specified in condition-name conditions.

The PROGRAM COLLATING SEQUENCE clause also applies to any nonnumeric merge or sort keys, unless the COLLATING SEQUENCE phrase is specified in the MERGE or SORT statement. When the PROGRAM COLLATING SEQUENCE clause is omitted, the EBCDIC collating sequence is used.

See Appendix D, "EBCDIC and ASCII Collating Sequences" on page 491 for more information about these sequences.

SEGMENT-LIMIT IS
Determines which segments will be considered as permanent segments of the object program.

segment-number
Must be an integer varying in value from 1 through 49.

When the SEGMENT-LIMIT 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.
- Those segments having segment-numbers from the segment-limit through 49 are considered as overlayable fixed segments.

For example, if SEGMENT-LIMIT IS 25 is specified, sections with segment-numbers 0 through 24 are permanent segments of the object program, and sections with segment-numbers 25 through 49 are overlayable fixed segments.

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.

SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph:

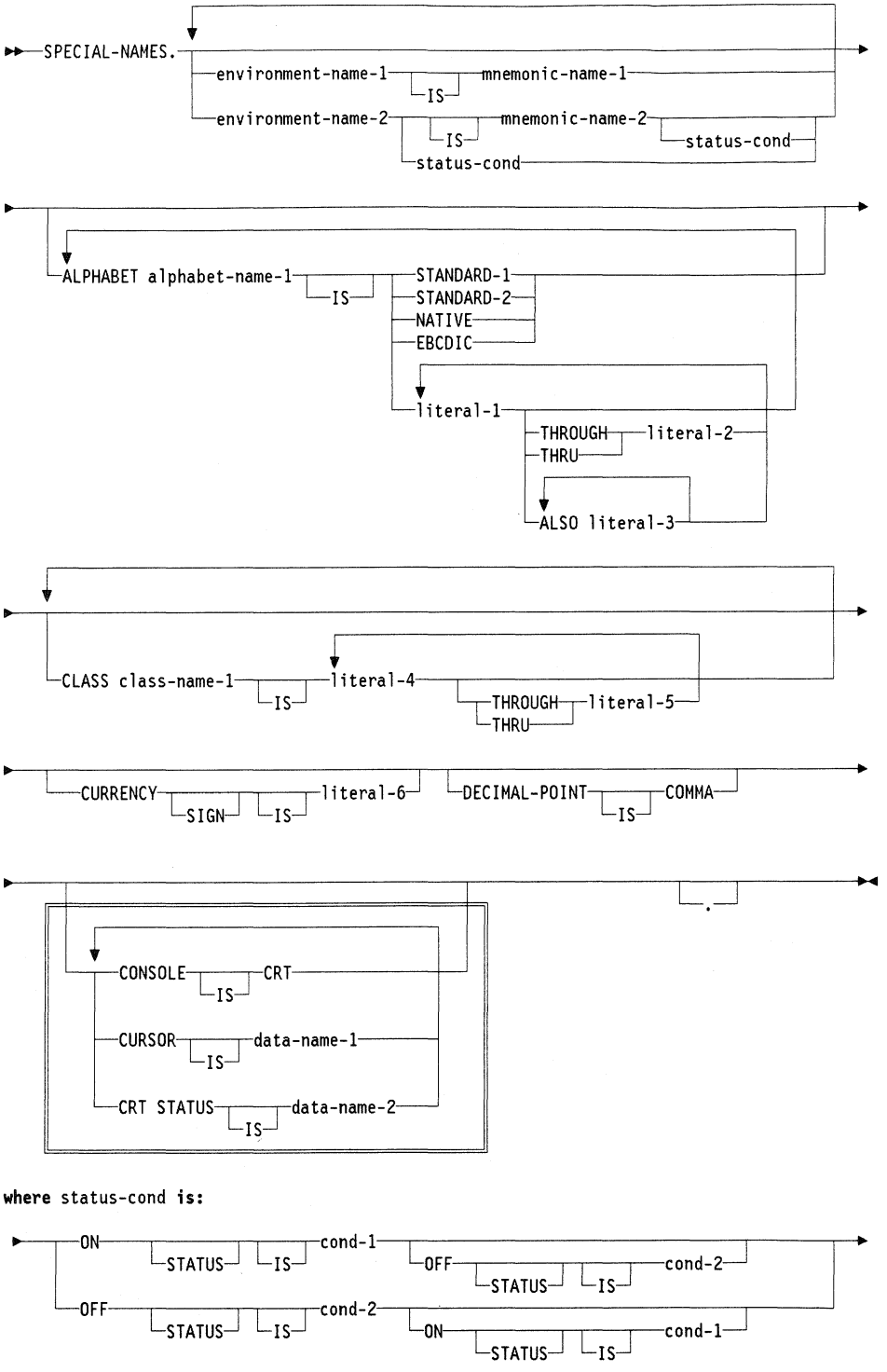
- Relates IBM-specified environment-names to user-defined mnemonic-names.
- Relates alphabet-names to character sets or collating sequences.
- Relates class names to sets of characters.
- Specifies a substitute character for the currency sign.
- Specifies that the functions of the comma and decimal point are to be interchanged in PICTURE clauses and numeric literals.

IBM Extension

- Specifies that ACCEPT or DISPLAY statements are treated as extended ACCEPT or DISPLAY statements.
- Specifies additional functions associated with ACCEPT statements.

End of IBM Extension

Format



Note: The optional separator period at the end of the format must be used if any of the optional clauses are selected.

environment-name-1

System devices or standard system actions taken by the compiler.

Table 3 shows the actions that are associated with mnemonic-names for environment-name-1.

Table 3. Choices of Environment-Name-1 and Action Taken

Environment-name-1	Statement where mnemonic-name associated with environment-name is used	Usage
CSP	WRITE	Suppress spacing when printing a line. Use only when PRINTER is the device.
C01	WRITE	Skip to the next page. Use only when PRINTER is the device.
ATTRIBUTE-DATA	ACCEPT	Retrieve attribute data about a program device acquired by a transaction file, but only when the file is open.
I-O-FEEDBACK	ACCEPT	Give information about the last I-O operation on a file, but only when the file is open.
OPEN-FEEDBACK	ACCEPT	Give information about a file, but only when the file is open.
CONSOLE, SYSTEM-CONSOLE	ACCEPT, DISPLAY	Communicate with the system operator's message queue (QSYSOPR).
LOCAL-DATA	ACCEPT, DISPLAY	Retrieve data from, or move data to the local data area created by the system for every job.
PIP-DATA	ACCEPT	Retrieve data from the Program Initialization Parameters (PIP) data area for programs running as part of a prestart job.[1]
REQUESTOR	ACCEPT, DISPLAY	Communicate with the user work station (interactive jobs) or the batch input stream or job log (batch jobs).
SYSIN	ACCEPT	The equivalent of REQUESTOR (for the ACCEPT statement only).
SYSOUT	DISPLAY	The equivalent of REQUESTOR (for the DISPLAY statement only).

[1] For more information regarding prestart jobs, see the *Work Management Guide*.

environment-name-2

Environment-name-2 can be defined as UPSI-0 through UPSI-7 or as SYSTEM-SHUTDOWN.

User Program Status Indicator (UPSI): Environment-name-2 can define eight one-byte program switches, UPSI-0 through UPSI-7.

Each UPSI is a User Program Status Indicator switch. UPSI-0 through UPSI-7 are COBOL names that identify program switches defined outside the COBOL program at object time. Their contents are considered to be alphanumeric. A value of zero is off; a value of one is on.

Each switch represents one byte from the 8-character SWS parameter of the control language CHGJOB, SBMJOB, JOB, and JOBD commands as follows:

UPSI-0 First byte (leftmost)
UPSI-1 Second byte
UPSI-2 Third byte
. .
. .
. .
UPSI-7 Eighth byte (rightmost)

SYSTEM-SHUTDOWN: SYSTEM-SHUTDOWN is an internal switch that is set to ON status when the system operator causes the system to be in a shutdown-pending state or when the job is being canceled in a controlled manner. The associated ON or OFF condition-names can be referenced anywhere a condition-name is valid. Their status cannot be altered by the program.

mnemonic-name-1
mnemonic-name-2

Mnemonic-name-1 and mnemonic-name-2 follow the rules of formation for user-defined names. Mnemonic-name-1 can be used in ACCEPT, DISPLAY, and WRITE statements. Mnemonic-name-2 can be referenced only in the SET statement. Mnemonic-name-2 can qualify cond-1 or cond-2 names.

ON STATUS IS, OFF STATUS IS

UPSI switches process special conditions within a program, such as year-beginning or year-ending processing. For example, at the beginning of the Procedure Division, an UPSI switch can be tested; if it is ON, the special branch is taken. (See "Switch-Status Condition" on page 199.)

cond-1, cond-2

Condition-names follow the rules for user-defined names. At least one character must be alphabetic. The value associated with the condition-name is considered to be alphanumeric. A condition-name may be associated with the on status and/or off status of each UPSI switch specified.

In the Procedure Division, the UPSI switch status is tested through the associated condition-name. Each condition-name is the equivalent of a level-88 item; the associated mnemonic-name, if specified, is considered the conditional variable and can be used for qualification.

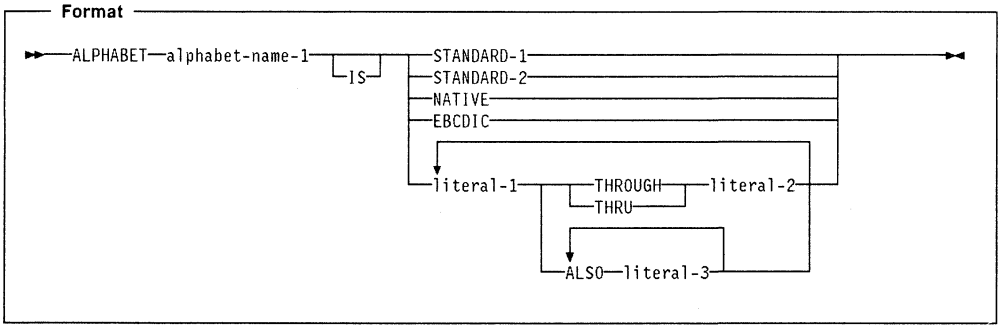
Coding Example

This coding example assigns mnemonic-names to some commonly used environment-names in the SPECIAL-NAMES paragraph.

```
SPECIAL-NAMES.  SYSTEM-CONSOLE IS SYSTM
    REQUESTOR IS WORK-STATION
    C01 IS NEXT-PAGE
    LOCAL-DATA IS LOCAL-DATA-AREA
    ATTRIBUTE-DATA IS ATTRB-DATA
    SYSTEM-SHUTDOWN IS SHUTDOWN-SWITCH
        ON STATUS IS SHUTDOWN-PENDING
    UPSI-0 IS UPSI-SWITCH-0
        ON STATUS IS U0-ON
        OFF STATUS IS U0-OFF
    UPSI-1 IS UPSI-SWITCH-1
        ON STATUS IS U1-ON
        OFF STATUS IS U1-OFF
    IBM-ASCII IS STANDARD-1
    CURRENCY-SIGN IS "Y".
```

ALPHABET Clause

Format



ALPHABET alphabet-name-1 IS

Provides a means of relating an alphabet-name to a specified character code set or collating sequence.

It specifies a **collating sequence** when used in either:

- The PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph
- The COLLATING SEQUENCE phrase of the SORT or MERGE statement.

It specifies a **character code set** when specified in the FD entry CODE-SET clause.

The EBCDIC collating sequence is used when the alphabet-name clause is omitted.

STANDARD-1

Specifies the ASCII character set.

STANDARD-2

Specifies the International Reference Version of the ISO 7-bit code defined in International Standard 646, 7-bit Coded Character Set for Information Processing Interchange.

NATIVE
Specifies the native character code set. This is the equivalent to EBCDIC for the OS/400 operating system.

EBCDIC
Specifies the EBCDIC character set.

literal-1, literal-2, literal-3
Specifies that the collating sequence is to be determined by the program, according to the following rules:

- The order in which literals appear specifies the ordinal number, in ascending sequence, of the character(s) in this collating sequence.
- Each numeric literal specified must be an unsigned integer and must have a value from 1 through 256 (the maximum number of characters in the EBCDIC character set). The value of each literal specifies the relative position of a character within the EBCDIC character set. For example:

  literal 112 represents the EBCDIC character ?
  literal 234 represents the EBCDIC character Z
  literal 241 represents the EBCDIC numeric character 0.

  Appendix D, "EBCDIC and ASCII Collating Sequences" on page 491, lists the ordinal number for each character in the EBCDIC and ASCII collating sequences.
- Each character in a nonnumeric literal represents that actual character in the EBCDIC character set. (If the nonnumeric literal contains more than one character, each character, starting with the leftmost, is assigned a successively ascending position within this collating sequence.)
- Any EBCDIC characters not explicitly specified assume positions in this collating sequence higher than any of the explicitly specified characters. The relative order of the unspecified characters remains unchanged from the EBCDIC collating sequence.
- Within one alphabet-name clause, a given character must not be specified more than once.
- Each nonnumeric literal associated with a THROUGH or ALSO phrase must be 1 character in length.
- When the THROUGH phrase is specified, the contiguous EBCDIC characters beginning with the character specified by literal-1 and ending with the character specified by literal-2 are assigned successively ascending positions in this collating sequence. This sequence may be either ascending or descending within the original EBCDIC sequence. For example, if the characters Z through S are specified, then for this collating sequence the ascending values are: ZYXWVUTS.
- When the ALSO phrase is specified, the EBCDIC characters specified as literal-1, literal-3, and so on, are assigned to the same position in this collating sequence. For example, if you specify:

  ```
  "D" ALSO "N" ALSO "%"
  ```

  the characters D, N, and % are all considered to be in the same position in the collating sequence.

- If specified as literals in the SPECIAL-NAMES paragraph, the figurative constants HIGH-VALUE and LOW-VALUE are associated with hex FF and hex 00 respectively.
- After all clauses in the SPECIAL-NAMES paragraph are processed, the character having the **highest** ordinal position in this collating sequence is associated with the figurative constant HIGH-VALUE. If more than one character has the highest position, because of specification of the ALSO phrase, the last character specified (or defaulted to when some characters in the native collating sequence are not explicitly specified) is considered to be the HIGH-VALUE character for procedural statements such as DISPLAY, or as the sending field in a MOVE statement. (If all characters within the native collating sequence were explicitly specified, and the ALSO phrase example from above were specified as the high-order characters of this collating sequence, the HIGH-VALUE character would be %.)
- After all clauses in the SPECIAL-NAMES paragraph are processed, the character having the **lowest** ordinal position in this collating sequence is associated with the figurative constant LOW-VALUE. If more than one character has the lowest position, because of specification of the ALSO phrase, the first character specified is the LOW-VALUE character. (If the ALSO phrase example given above were specified as the low-order characters of the collating sequence, the LOW-VALUE character would be D.)

When **literal-1**, **literal-2**, or **literal-3** is specified, the alphabet-name must **not** be referred to in a CODE-SET clause (see "CODE-SET Clause" on page 123).

Alphabet-Name Clause Examples: The following examples illustrate some uses for the alphabet-name clause.

If PROGRAM COLLATING SEQUENCE IS USER-SEQUENCE; if the alphabet-name clause is specified as USER-SEQUENCE IS "D", "E", "F"; and if two Data Division items are defined as follows:

```
01 ITEM-1 PIC X(3) VALUE "ABC".
01 ITEM-2 PIC X(3) VALUE "DEF".
```

then the following comparison is true:

```
IF ITEM-1 > ITEM-2
```

Characters D, E, and F are in ordinal positions 1, 2, and 3 of this collating sequence. Characters A, B, and C are in ordinal positions 197, 198, and 199 of this collating sequence.

If the alphabet-name clause is USER-SEQUENCE IS 1 THRU 247, 251 THRU 256, "7", ALSO "8", ALSO "9"; if all 256 EBCDIC characters have been specified; and if the two Data Division items are specified as follows:

```
01 ITEM-1 PIC X(3) VALUE HIGH-VALUE.
01 ITEM-2 PIC X(3) VALUE "787".
```

then both of the following comparisons are true:

```
IF ITEM-1 = ITEM-2 . . .
IF ITEM-2 = HIGH-VALUE . . .
```

They compare as true because the values "7", "8", and "9" all occupy the same position (HIGH-VALUE) in this USER-SEQUENCE collating sequence.

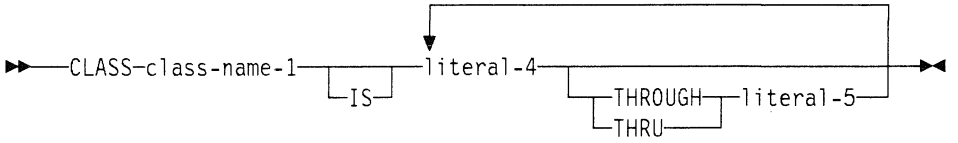
If the alphabet-name clause is specified as USER-SEQUENCE IS "E", "D", "F" and a table in the Data Division is defined as follows:

```
05 TABLE A OCCURS 6 ASCENDING KEY IS
   KEY-A INDEXED BY INX-A.
   10 FIELD-A ...
   10 KEY-A ...
```

and if the contents in ascending sequence of each occurrence of KEY-A are A, B, C, D, E, G, then the results of the execution of a SEARCH ALL statement for this table will be invalid because the contents of KEY-A are not in ascending order. The proper ascending order would be E, D, A, B, C, G.

CLASS Clause

Format



CLASS class-name-1 IS

Provides a means for relating a name to the specified set of characters listed in that clause. Class-name 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 consists.

literal-4, literal-5

If numeric, must be unsigned integers and must have a value from 1 through 256 (the maximum number of characters in the EBCDIC character set).

The value of each literal specifies the relative position, or ordinal number, of a character within the EBCDIC character set. Appendix D, "EBCDIC and ASCII Collating Sequences" on page 491 lists the ordinal number for each character in the EBCDIC collating sequence.

If nonnumeric, the literal is the actual character within the EBCDIC character set. If the value of the nonnumeric literal contains multiple characters, each character in the literal is included in the set of characters identified by class-name.

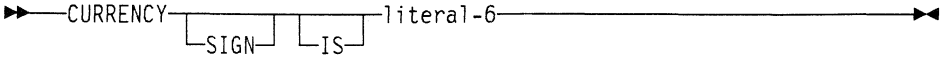
If the nonnumeric literal is associated with a THROUGH phrase, it must be one character in length.

THROUGH, THRU

THROUGH and THRU are equivalent. If THROUGH is specified, class-name includes those characters beginning with the value of literal-4 and ending with the value of literal-5. In addition, the characters specified by a THROUGH phrase may specify characters in either ascending or descending order.

CURRENCY SIGN Clause

Format



CURRENCY SIGN IS
Currency symbol in the PICTURE clause.

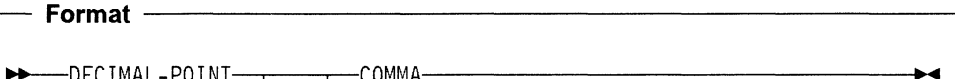
literal-6
Must be a one-character, nonnumeric literal, and must **not** be any of the following:

- Digits zero (0) through nine (9)
- Uppercase alphabetic characters A B C D P R S V X Z
- Lowercase alphabetic characters a through z
- The space
- Special characters * + - / , . ; ( ) = ".
- A figurative constant.

When the CURRENCY SIGN clause is omitted, only the dollar sign ($) may be used as the PICTURE symbol for the currency sign.

DECIMAL-POINT Clause

Format

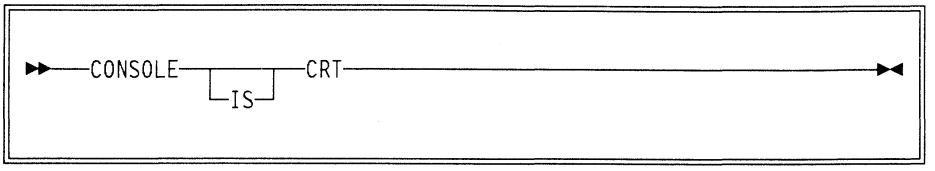


Exchanges the functions of the period and the comma in PICTURE character strings and in numeric literals.

IBM Extension

CONSOLE Clause

Format



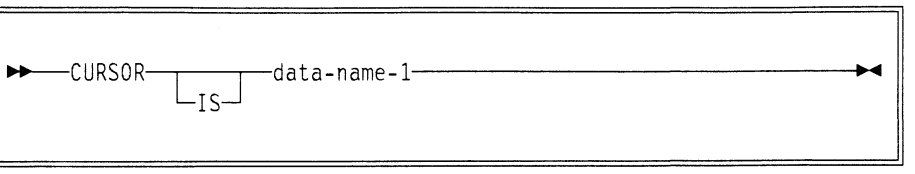
If CONSOLE IS CRT is present, any ACCEPT or DISPLAY statement that has no phrases specific to a particular format (such as LOCAL-DATA or PIP-DATA), is treated as an extended ACCEPT or DISPLAY statement.

If CONSOLE IS CRT is not present, any ACCEPT or DISPLAY statement that has no phrases specific to a particular format is treated as a standard ANSI X3.23-1985 COBOL ACCEPT or DISPLAY statement.

See the "Workstation I/O" on page 228 and the "Format 3 — Extended DISPLAY Statement" on page 278 for descriptions of the conditions which determine whether ACCEPT or DISPLAY statements are extended or standard.

CURSOR Clause

Format



CURSOR IS data-name-1

Specifies the data item that will contain the cursor address used by the ACCEPT statement.

data-name-1

Must be a 4- or 6-byte alphanumeric field or a 4- or 6-byte unsigned zoned integer field. If data-name-1 is 4 characters in length, the first two characters are interpreted as line number, and the second two as column number. If data-name-1 is 6 characters in length, the first three characters are interpreted as line number, and the second three as column number.

The clause has no effect if data-name-1 contains an invalid position value (such as zeros, a nonnumeric value, or a value that is beyond the range of the screen).

Data-name-1 must be declared in the WORKING-STORAGE SECTION of the program.

CURSOR Clause Considerations

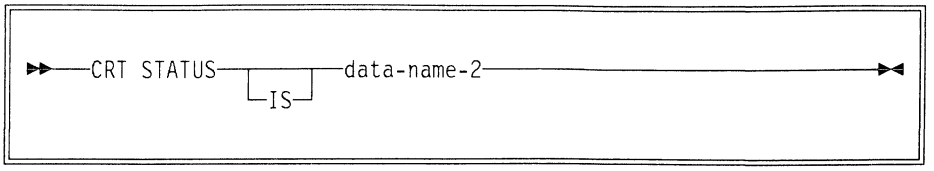
At the start of an extended ACCEPT operation, if data-name-1 contains a value that is a valid character position on the screen, that position is used as the initial position for the cursor. A valid position is a coordinate that lies on the screen (that is, within the range from line 1, column 1, to line 24, column 80). After the ACCEPT operation, if the position in data-name-1 was valid, data-name-1 is updated to show the position of the cursor at the end of the operation.

If the CURSOR IS identifier contains an invalid value (such as spaces, low-values, high-values or a value outside of the screen range), the cursor is positioned at the start of the first input field that is active on the screen.

CURSOR IS has no effect on the positioning of fields on the screen.

CRT STATUS Clause

Format



CRT STATUS IS data-name-2
Specifies a data item into which a status value is moved after an extended ACCEPT statement.

data-name-2
Must be described in the WORKING-STORAGE SECTION and must be a 6-byte alphanumeric field or a 6-byte unsigned zoned integer.

CRT STATUS Clause Considerations

If the CRT STATUS clause is specified in the SPECIAL-NAMES paragraph, every extended ACCEPT statement places a value into data-name-2 to indicate the outcome of the ACCEPT operation. Data-name-2 consists of status keys which are set to indicate possible conditions resulting from the completion of the operation.

CRT Status Key 1: The first two bytes of data-name-2 form CRT Status Key 1 and should be described as PIC 99. It indicates the condition that caused the termination of the ACCEPT operation. The possible values are:

0 Indicates a terminating key such as an enter key, or an auto skip from the final field

1 Indicates a function key

9 Indicates an error

A termination that returns a value of 0 is a normal completion (normal termination).

If the ACCEPT statement contains an ON EXCEPTION phrase, any value in CRT Status Key 1, except 0, will cause the execution of the imperative statement in the ON EXCEPTION phrase.

CRT Status Key 2: The next two bytes of data-name-2 form CRT Status Key 2, and contain a code giving further details of the condition that terminated the ACCEPT operation. Its format and possible values depend on the value in CRT Status Key 1, as shown in the following table.

Table 4. Valid Combinations of CRT STATUS Keys 1 and 2

KEY 1	KEY 2		Meaning
	Format	Value	
0	PIC 99	0	The operator pressed a terminating key
0	PIC 99	1	Auto skip from the last field[1]
1	PIC 99	1-24	The function key number
9	PIC 99	0	Error condition (no items fall within the screen)

Note: [1] When auto skip from the last field takes place, the value of 1 for CRT STATUS KEY 2 is returned to supported controllers, and the value of 0 is returned to those controllers not supported. This relationship is shown in Table 5.

Table 5. Auto Skip Value Returned by Controller Type

Type of Controller	Auto Skip Value of 1 Returned
AS/400 controllers:	
Local workstation controllers	Yes
Remote 5251 model 12	Not applicable
Remote 5294	No
Remote 5394	Yes, if installed with new workstation controller code
Remote 3174	No, with *NOUNDSPCHR option
Remote 3274	No, with *NOUNDSPCHR option
PC attachments:	
DOS and OS/2* operating environments	No
System to system passthru:	
AS/400 system to AS/400 system	Yes
System/36 to AS/400 system	No
System/38 to AS/400 system	No

CRT Status Key 3: The last two bytes of data-name-2 form CRT Status Key 3. If CRT Status Key 1 is 0, CRT Status Key 3 contains the code for the keyboard key that terminated the ACCEPT operation. Otherwise, if CRT Status Key 1 is 9, an error is signaled by the operating system, and CRT Status Key 3 will be set to 99.

The codes for the keys are:

- 00 Enter key
- 90 Roll up key
- 91 Roll down key
- 92 Print key
- 93 Help key
- 94 Clear key
- 95 Home key

End of IBM Extension

Environment Division—Input-Output Section

The Input-Output Section defines each file, identifies its external storage medium, assigns the file to one or more input/output devices, and also specifies information needed for efficient transmission of data between the external medium and the COBOL program.

File Categories

The AS/400 system has four categories of files: database files, device files, DDM files, and save files.

This manual uses the term *file* to mean any of these files.

Database Files

Database files allow information to be permanently stored on the system. Multiple programs can access this information in different ways.

A database file is subdivided into groups of records called members.

There are two types of database files: physical files and logical files.

Physical Files: A physical file is a file that actually contains data records. This makes physical files similar to disk files on other systems. A physical file can contain only fixed-length records, all of which have the same format.

Logical Files: A logical file is a database file through which data from one or more physical files can be accessed. The format and organization of this data is different from that of the data in the physical file(s). Each logical file can define a different access path (index) for the data in the physical file(s). Each logical file can exclude and reorder the fields defined in the physical file(s).

Device Files

A device file reads from or writes to a device or remote system. A device file controls the transfer of data between the physical device or remote system and the program.

DDM Files

Distributed Data Management (DDM) allows you to access data that reside on remote systems that support DDM. DDM files are supported by the COBOL compiler. You can retrieve, add, update, or delete data records in a file that resides on another system.

When you compile a source program that is on a remote system, the COBOL/400 compiler expects a source type of CBL. If the source type is not CBL, the compiler issues a message indicating that it encountered an unexpected source member type. To resolve this discrepancy, you should recompile the program in the environment indicated by the source member type, or change the source member type, or use the correct compiler indicated by the source member type.

For more information about accessing remote files, refer to the *DDM Guide*.

Save Files

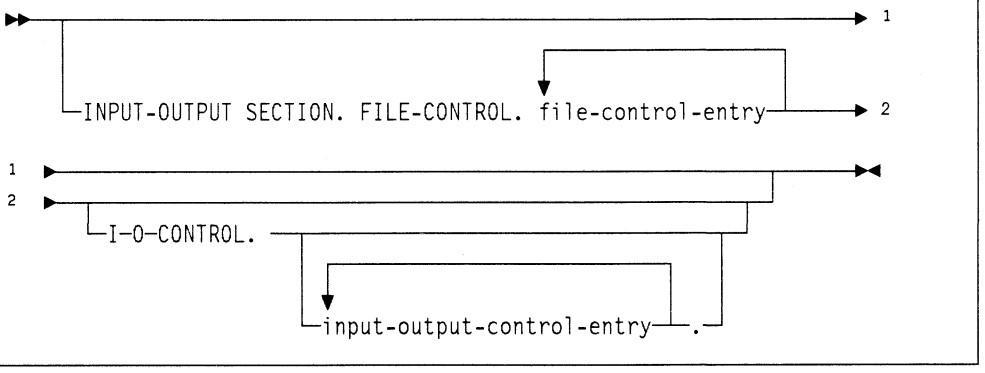
A save file is a file that is used to prepare data in a format that is correct for backup and recovery purposes or for transportation to another system. A save file can be used to contain the output that is produced from the Save Library (SAVLIB) or Save Object (SAVOBJ) CL commands. See the *Data Management Guide* for information about save files.

Paragraphs

The Input-Output section of the Environment Division contains two paragraphs:

- FILE-CONTROL paragraph
- I-O-CONTROL paragraph.

Format



FILE-CONTROL paragraph

Names and associates the files with the external media.

The key word FILE-CONTROL may appear only once, at the beginning of the FILE-CONTROL paragraph. It must begin in Area A, and be followed by a separator period.

file-control-entry

Must begin in Area B with a SELECT clause. It must end with a separator period. See "FILE-CONTROL Paragraph" on page 72.

I-O-CONTROL paragraph

Specifies information needed for efficient transmission of data between external media and the COBOL program.

input-output-control-entry

The series of entries must end with a separator period. See "I-O-CONTROL Paragraph" on page 90.

The exact contents of the Input-Output Section depend on the file organization and access methods used. See "ORGANIZATION Clause" on page 80 and "ACCESS MODE Clause" on page 81.

FILE-CONTROL Paragraph

The FILE-CONTROL paragraph associates each file in the COBOL program with an external medium, and specifies file organization, access mode, and other information.

COBOL allows for four distinct kinds of file input and output:

- Sequential
- Indexed
- Relative

IBM Extension

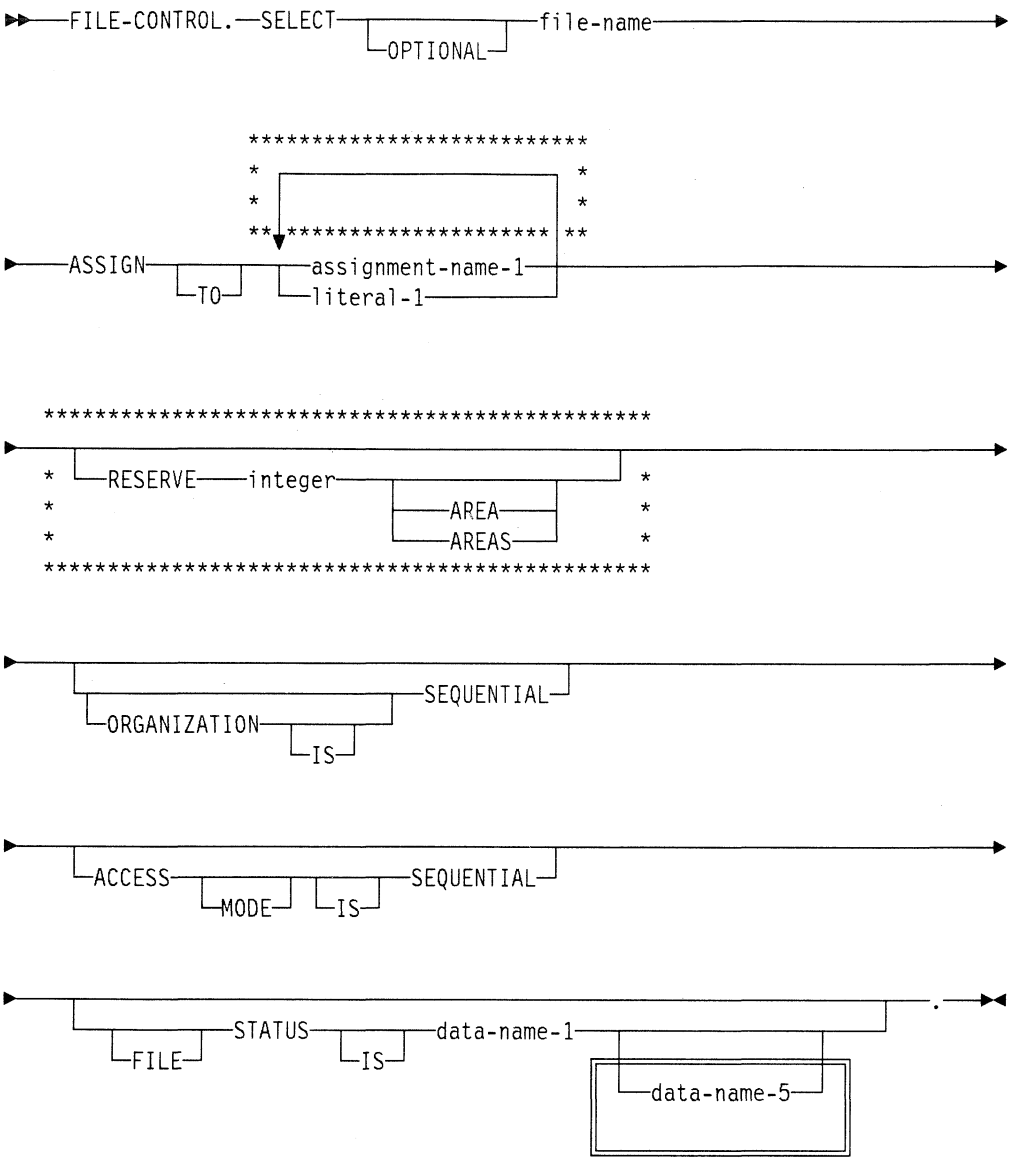
- Transaction

End of IBM Extension

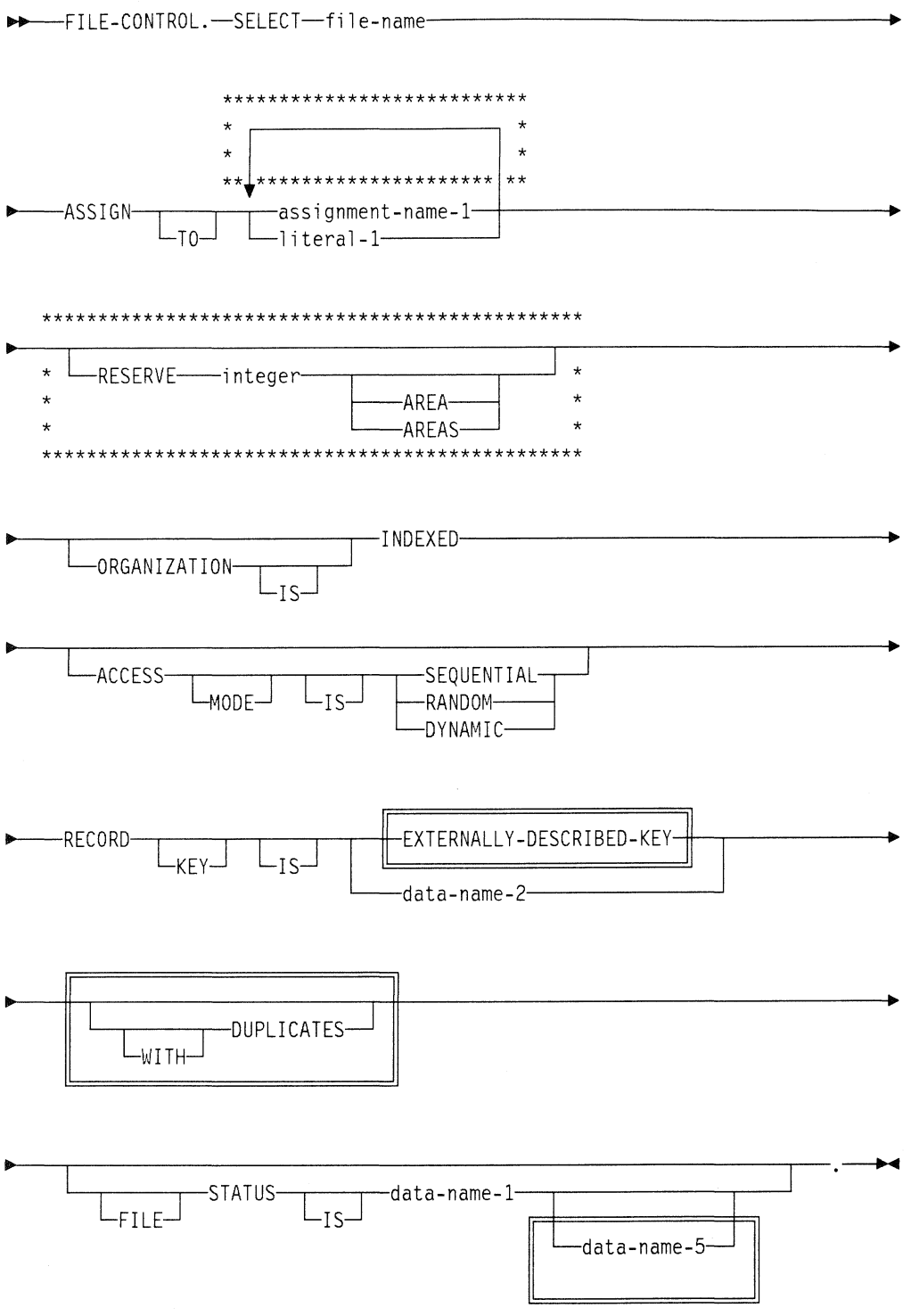
The FILE-CONTROL paragraph begins with the word "FILE-CONTROL", followed by a separator period. It must contain one and only one entry for each file described in an FD or SD entry in the Data Division. Within each entry, the SELECT clause must appear first. The other clauses may appear in any order.

Each data-name must appear in a Data Division data description entry. Each data-name can be qualified but cannot be subscripted or indexed.

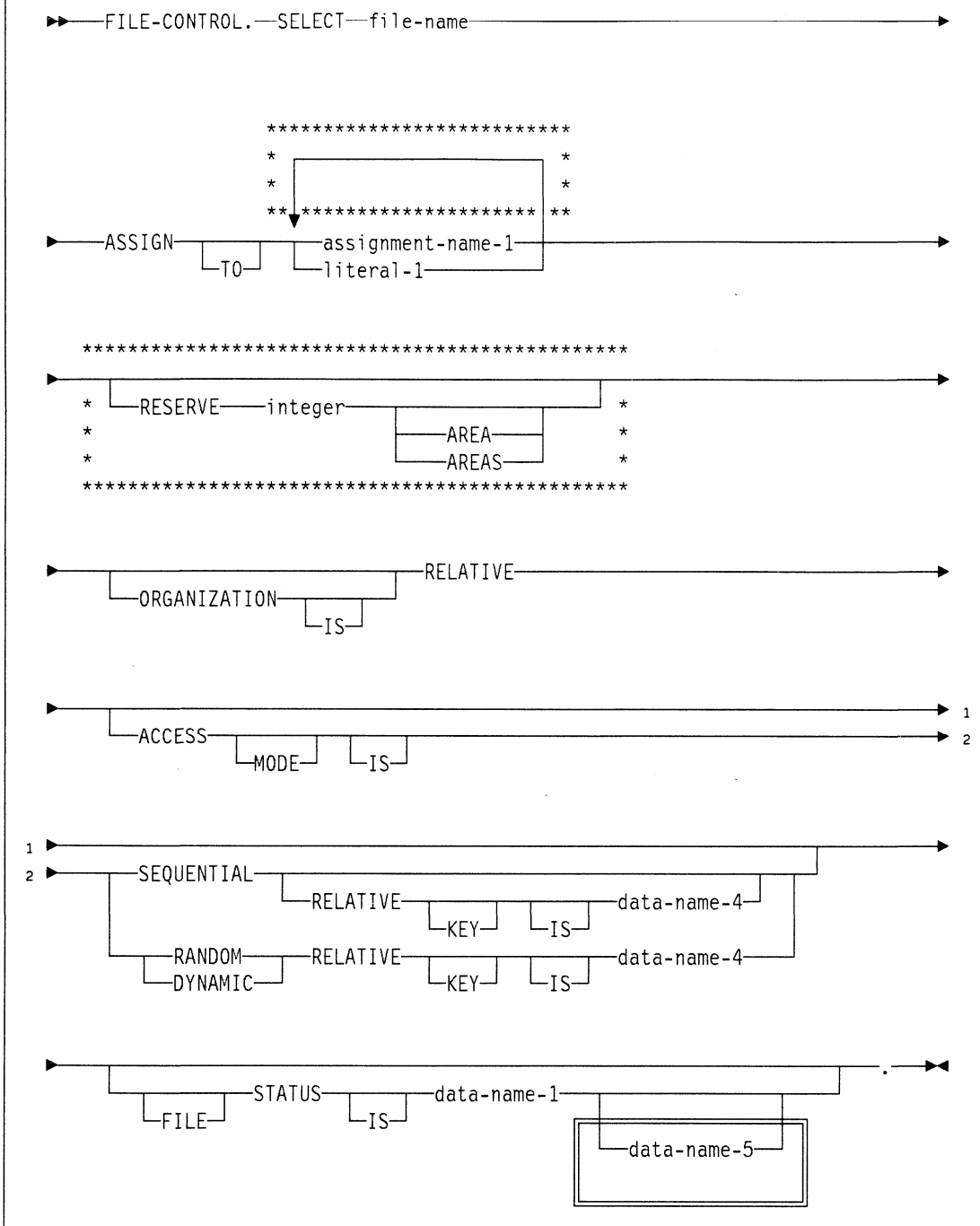
Format 1 - Sequential



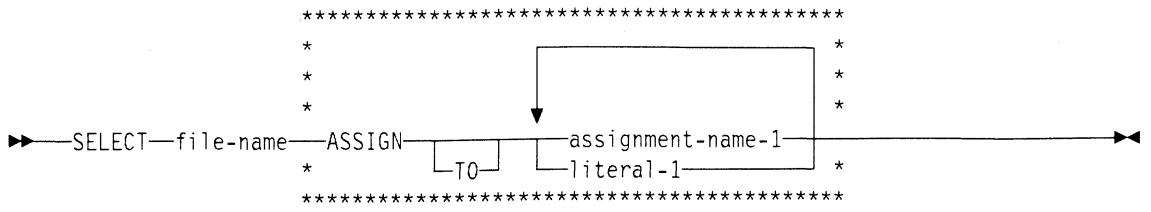
Format 2 - Indexed



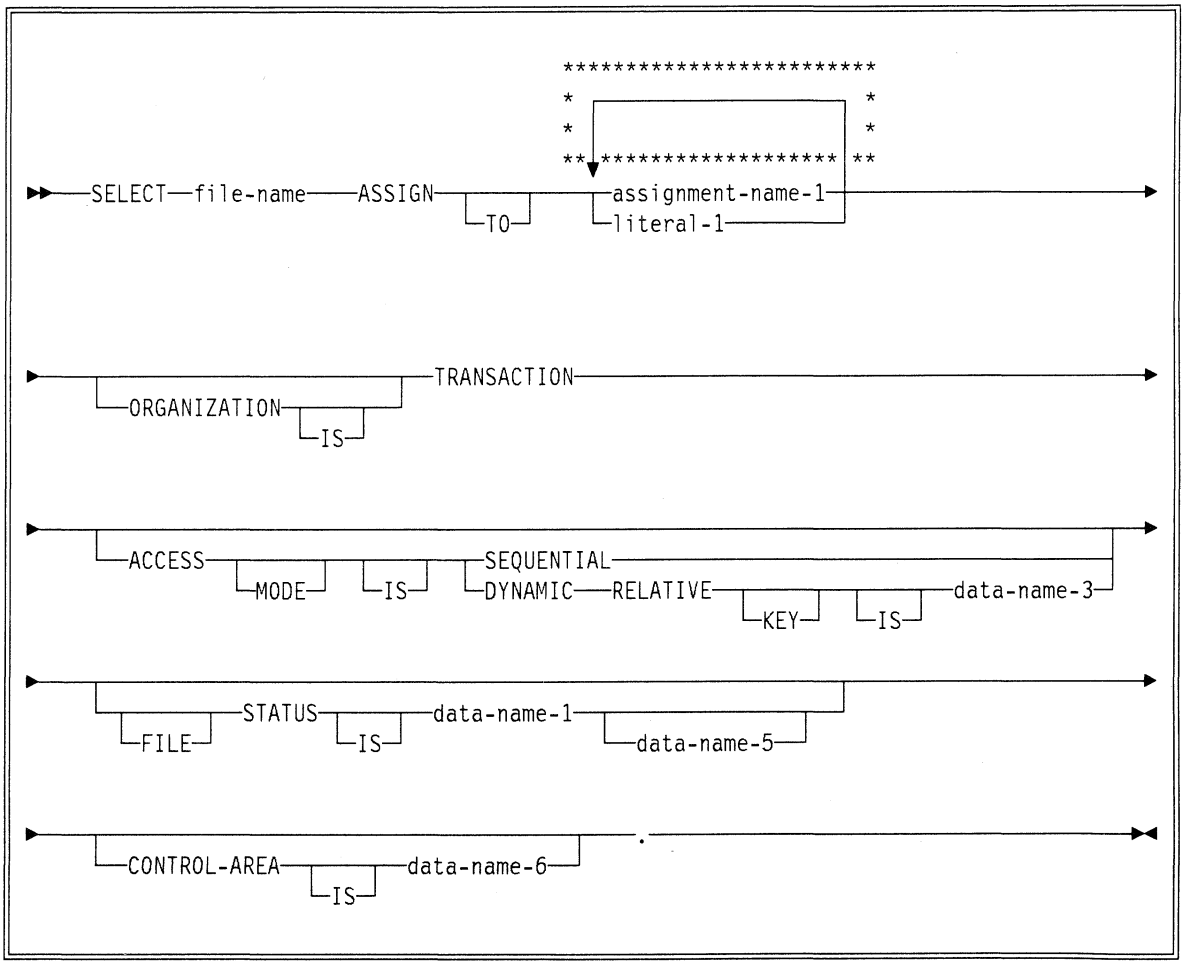
Format 3 - Relative



Format 4 - Sort or Merge



Format 5 - Transaction



See the chapter on Transaction Files in the *COBOL/400 User's Guide* for more information on working with transaction files.

SELECT Clause

The SELECT clause chooses a file.

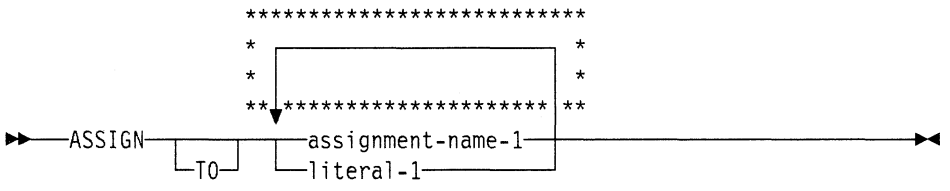
SELECT OPTIONAL (Format 1)
May be specified only for sequential files opened in the input, I-O, or extend mode. You must specify SELECT OPTIONAL for such input files that are not necessarily present each time the object program is executed.

file-name
Must be identified by an FD or SD entry in the Data Division. A file-name must conform to the rules for a COBOL user-defined name, must contain at least one alphabetic character, and must be unique within this program.

When file-name specifies a sort or a merge file, only the ASSIGN clause may follow the SELECT clause.

ASSIGN Clause

Format



The ASSIGN clause associates a file with an external medium.

For sort or merge files (associated with an SD entry), no external medium is used. The related ASSIGN clause is only validity checked. It is not actually used for I-O.

assignment-name-1
literal-1
The assignment-name or literal makes the association between the file and the external medium.

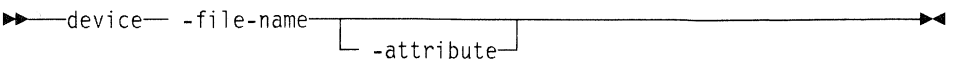
Any assignment-name or literal after the first is syntax checked, but has no effect on the execution of the program

Assignment-name-1 or literal-1 consists of 3 parts:

- Device
- File name
- Attribute.

It has the following general structure:

Format



Device: This part specifies the type of device that the file will use. The compiler can then check whether the file is described and used in a consistent manner. See the *COBOL/400 User's Guide* for further information.

Notes:

1. The compiler does not check whether the device associated with the external file is of the type specified in the device portion of assignment-name-1 or literal-1.
2. The compiler provides no diagnostics unless the I-O verbs were used in an inconsistent manner.
3. When the program runs, the operating system could either issue an escape message or ignore the function if it was not applicable to the device. See the *Data Management Guide* for further information on overriding files.

IBM Extension

The device that the file will use can be changed at run time with the OVRxxxF CL command. To ensure consistent results, the device type associated with the file should correspond to that given in the assignment-name.

End of IBM Extension

Device can be any of the following:

Device	Associated File
PRINTER[1]	Printer file
FORMATFILE[2] [5]	Printer file
TAPEFILE	Tape file
DISKETTE	Diskette file
DISK[3]	Any physical database file or single format logical database file
DATABASE[4] [5]	Any database file (or DDM file)
WORKSTATION[5]	Display file or ICF file

[1] PRINTER should be specified for program described printer files only.

[2] FORMATFILE should be specified for externally described printer files only. For more information on how to use externally described printer files see the section on FORMATFILE Files in the *COBOL/400 User's Guide*.

[3] When DISK is the device, database extensions cannot be used. See the *COBOL/400 User's Guide* for more information about DISK files.

[4] When DATABASE is the device, externally described data and database extensions can be used. However these file types are not supported for dynamic file creation. See the "OPEN Statement" on page 329 for information about Dynamic Files. See the *COBOL/400 User's Guide* for more information about DATABASE files.

[5] See the "OPEN Statement" on page 329 for information about Dynamic Files.

File Name: This part of assignment-name must be an unhyphenated, 1- through 10-character system name of the actual external file (physical or logical database, or device). This external file has to be created before compiling the program only when it is used by a COPY statement, DDS (data description specifications) or DD format, within this program.

For database files, the member name cannot be specified in the program. If a member other than the first member is to be specified, the Override with Database File (OVRDBF) CL command must be used at execution time to specify the member name.

This file name is the name of the AS/400 object that is displayed by the Display Program References (DSPPGMREF) command. Since no external medium is used for an SD file, the DSPPGMREF command does not list any files defined for an SD file.

The file name can be changed at execution time with the TOFILE parameter of the OVRxxxF CL command. To ensure consistent results, the device type associated with the TOFILE parameter should be the same as that specified for assignment-name-1 or literal-1.

Attribute: This part of assignment-name-1 or literal-1 can be *SI*. *SI* indicates that a separate indicator area has been specified in the DDS for a FORMATFILE or WORKSTATION file.

See the *COBOL/400 User's Guide* for details on the use of the SI attribute and further information about the ASSIGN clause.

The valid entries for each field of assignment-name-1 or literal-1 vary with the device. The valid combinations of fields are shown in Figure 4.

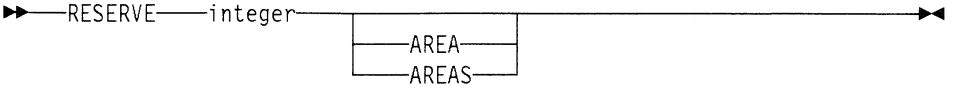
Device	File Name	Default File Name	SI
PRINTER	O	QPRINT	N
FORMATFILE	R		O
TAPEFILE	O	QTAPE	N
DISKETTE	O	QDKT	N
DISK	R		N
DATABASE	R		N
WORKSTATION	R		O

R = Required
O = Optional
N = Not Allowed

Figure 4. Valid Entries for Assignment-Name-1 and Literal-1

RESERVE Clause

Format



The RESERVE clause reserves input-output areas.

The RESERVE clause is syntax-checked, but is treated as documentation.

ORGANIZATION Clause

The ORGANIZATION clause specifies the logical structure of the file. The file organization is established at the time the file is created and cannot subsequently be changed. When the ORGANIZATION clause is omitted, ORGANIZATION IS SEQUENTIAL is assumed.

IBM Extension

For database files, the ORGANIZATION clause indicates the current program usage of the file in the program. Therefore, the same database file can use SEQUENTIAL, INDEXED (assuming a keyed sequence access path exists), or RELATIVE in the ORGANIZATION clause. This is true regardless of what is specified in other programs that use this file.

Note: A keyed sequence access path is always created when a key is specified in the DDS that was used as input to the Create Physical File (CRTPF) or the Create Logical File (CRTLF) CL command.

End of IBM Extension

ORGANIZATION IS SEQUENTIAL (Format 1): A predecessor-successor relationship of the records in the files is established by the order in which records are placed in the file when it is created or extended (arrival sequence access path).

ORGANIZATION IS INDEXED (Format 2): The position of each logical record in the file is determined by the key sequence access path created with the file and maintained by the system. The access path is based on an embedded key within the file's records.

ORGANIZATION IS RELATIVE (Format 3): The position of each record in the file is determined by its relative record number within the arrival sequence access path.

IBM Extension

ORGANIZATION IS TRANSACTION (Format 5): Signifies interaction between a COBOL program and either a work station user or another system. For more information on transaction files, see the *COBOL/400 User's Guide*.

End of IBM Extension

ACCESS MODE Clause

The ACCESS MODE clause defines the manner in which the records of the file are made available for processing. If the ACCESS MODE clause is not specified, sequential access is assumed.

ACCESS MODE IS SEQUENTIAL

Can be specified in all three formats.

Format 1—Sequential

Records in the file are accessed in the sequence established when the file was created or extended (arrival sequence). Format 1 supports only sequential access.

Format 2—Indexed

Records in the file are accessed in the sequence of ascending record key values according to the collating sequence of the file.

IBM Extension

When using an externally described file, if the DDS keyword DESCEND is used when the field is specified as a key field, the records in the file are accessed in the sequence of descending record key values within the index. Either the DESCEND keyword, or the ASCEND keyword (if DESCEND is not specified) appears under the heading RETRIEVAL in a comment table in the COBOL source program listing.

End of IBM Extension

Format 3—Relative
Records in the file are accessed in the ascending sequence of relative record numbers of existing records in the file.

ACCESS MODE IS RANDOM
Can be specified in Formats 2 and 3 only.

Format 2—Indexed
The value placed in a record key data item specifies the record to be accessed.

Format 3—Relative
The value placed in a relative key data item specifies the record to be accessed.

ACCESS MODE IS DYNAMIC
Can be specified in Formats 2 and 3 only.

Format 2—Indexed
Records in the file may be accessed sequentially or randomly, depending on the form of the specific input-output request.

Format 3—Relative
Records in the file may be accessed sequentially or randomly, depending on the form of the specific input-output request.

Data Organization and Access Modes

Data organization is the permanent logical structure of the file. You tell the computer how to retrieve records from the file by specifying the **access mode**. In COBOL you can specify any of four types of data organization, and three access modes. The *COBOL/400* Reference Summary* shows which clauses and statements are required, and which clauses and statements are optional for each access mode and device. See the *COBOL/400 User's Guide* for information about COBOL file processing in relation to AS/400 file processing.

Note: Sequentially organized data may only be accessed sequentially; however, data that has indexed or relative organization may be accessed with any of the three access methods.

Data Organization

In a COBOL program, data organization can be sequential, indexed, relative, or TRANSACTION.

Records can be fixed or variable in length. For all files other than tape, variable length records are stored as fixed length records of the maximum size specified for the file.

Sequential Organization: The physical order in which the records are placed in the file determines the sequence of records. The relationships among records in the file do not change, except that the file can be extended. There are no keys. Both database files and device files can have sequential organization.

Each record in the file, except the first, has a unique predecessor record, and each record, except the last, also has a unique successor record.

Indexed Organization: Each record in the file has an embedded key called a key data item that is associated with an index. An index provides a logical path to the data records, according to the contents of the associated embedded record key data items. Only database files can have indexed organization.

Each record in an indexed file must have an embedded prime key data item. When records are inserted, updated, or deleted, they are identified solely by the values of their prime keys. Thus, the value in each prime key data item must be unique and must not be changed when the record is updated. You tell COBOL the name of the prime key data item on the RECORD KEY clause of the FILE-CONTROL paragraph.

IBM Extension

A logical file that is opened for OUTPUT does not remove all records in the physical file on which it is based. Instead, the file is opened to allow only write operations, and the records are added to the file.

End of IBM Extension

Relative Organization: Think of the file as a string of record areas, each of which contains a single record. Each record area is identified by a relative record number; the access method stores and retrieves a record, based on its relative record number. For example, the first record area is addressed by relative record number 1, and the 10th is addressed by relative record number 10. The physical sequence in which the records were placed in the file has no bearing on the record area in which they are stored, and thus on each record's relative record number. Relative files must be assigned to DISK or DATABASE.

Table 6 on page 84 summarizes conditions affecting relative output files.

Table 6. Initialization of Relative Output Files

File Access and CL Specifications	Conditions at Opening Time	Conditions at Closing Time	File Boundary
Sequential *INZDLT		Records not written are initialized	All increments
Sequential *INZDLT *NOMAX size		CLOSE succeeds File status is 0Q	Up to boundary of records written
Sequential *NOINZDLT			Up to boundary of records written
Random or dynamic	Records are initialized File is open		All increments
Random or dynamic *NOMAX size	OPEN fails File status is 9Q		File is empty

To extend a file boundary beyond the current number of records and within the file size, use the INZPFM command to add deleted records before processing the file. You will need to do this when more records need to be added to the file, and file status 0Q has been received.

Any attempt to extend a relative file beyond its current size results in a boundary violation.

To recover from a file status of 9Q, use the CHGPF command as described in the associated run-time message text.

Relative record number processing can be used for a physical file or for a logical file that is based on only one physical file.

IBM Extension

TRANSACTION Organization: Work station and data communication files can have TRANSACTION organization. See the Transaction Files chapter in the *COBOL/400 User's Guide* for a discussion of using this organization.

End of IBM Extension

Access Modes

Access mode is a COBOL term that defines the manner in which data in a logical or physical file is to be processed. The three access modes are sequential, random, and dynamic.

Sequential-Access Mode: Allows reading and writing records of a file in a serial manner; the order of reference is implicitly determined by the position of a record in the file.

Random-Access Mode: Allows reading and writing records in a programmer-specified manner; the control of successive references to the file is expressed by specifically defined keys supplied by the user.

Dynamic-Access Mode: Allows a specific input-output request to determine the access mode. Therefore, records may be processed sequentially and/or randomly.

Relationship Between Data Organizations and Access Modes

Sequential Files: Files with sequential organization can be accessed only sequentially. The sequence in which records are accessed is the order in which the records were originally written.

Indexed Files: All three access modes are allowed.

In the sequential access mode, the sequence in which records are accessed is determined by the prime record key value.

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

In the dynamic access mode, you may change, as necessary, from sequential access to random access, using appropriate forms of input-output statements.

Relative Files: All three access modes are allowed.

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

In the random access mode, you control the sequence in which records are accessed. The desired record is accessed by placing its relative record number in a RELATIVE KEY data item; the RELATIVE KEY must not be defined within the record description entry for this file.

In the dynamic access mode, you may change, as necessary, from sequential access to random access, using the appropriate forms of input-output statements.

IBM Extension

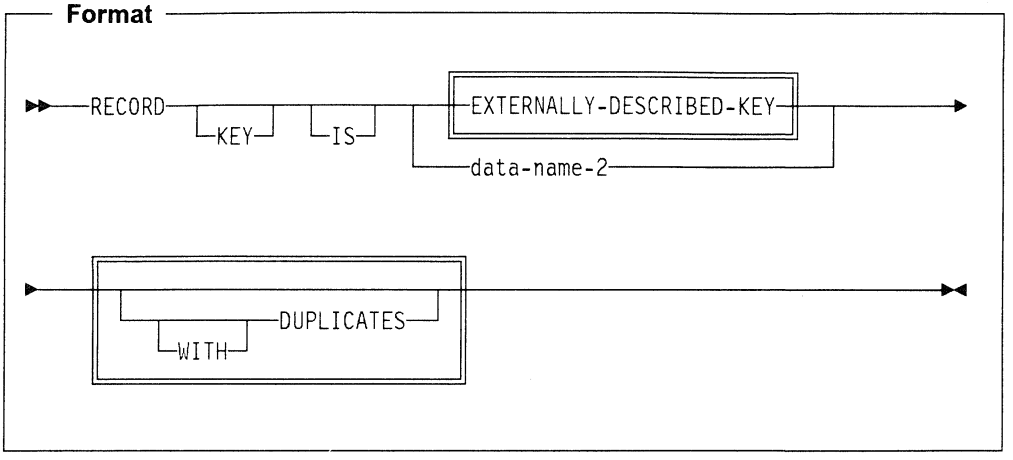
Transaction Files: See the Transaction Files chapter in the *COBOL/400 User's Guide* for a discussion of access mode considerations for transaction files.

End of IBM Extension

RECORD KEY Clause

The RECORD KEY clause (Format 2) must be specified for an indexed file. The RECORD KEY clause specifies the data item within the record that is the record key for an indexed file. Each record in the file must have a unique record key value.

Format



IBM Extension

The DUPLICATES phrase can only be specified for files assigned to DATABASE. This allows the file to have keys with the same values. If the file has multiple formats, two keys in different formats have the same values only when the key lengths and the contents of the keys are the same.

For example, given a file with the following two formats:

Format F1 with keys A, B, C
Format F2 with keys A, B, D.

If fields C and D are the same length, have the same data type, and have the same values, the file would contain two records with a duplicate key. The term *duplicate key* applies only to a complete record key for the format. A record key for the format consists of the key field(s) defined for a DDS format for records residing on the database. The term does not apply to the common key for the file (only fields A and B in the above example).

Users can indicate DUPLICATES on the RECORD KEY clause. A file status of 95 is returned after a successful open when:

- The DUPLICATES phrase is specified in the COBOL program and the file was created with UNIQUE specified in DDS.
- The DUPLICATES phrase is not specified in the COBOL program and the file was created allowing nonunique keys.

Processing files when either of these conditions exist can cause unpredictable results.

In a file that allows duplicates and is processed randomly or dynamically, the duplicate record that is updated or deleted must be the proper one. To ensure this, the last input/output statement processed prior to the REWRITE or DELETE

operation must be a successfully processed READ statement without the NO LOCK phrase.

If the DDS file level keyword LIFO (last-in-first-out) is specified, the duplicate records within a physical file are retrieved in a last-in-first-out order.

End of IBM Extension

data-name-2

Data-name-2 is the RECORD KEY data item. It must be described as a fixed-length alphanumeric item within a record description entry associated with the file. The length of the record key is restricted; the key length, in bytes, cannot exceed 2 000. See the *DDS Reference* for more information. Data-name-2 may be qualified, but it must not be subscripted.

IBM Extension

The RECORD KEY data item, data-name-2, can be a numeric item when the file is assigned to a DATABASE device type. The numeric item can have a usage of DISPLAY, COMP (COMP-3), COMP-4, PACKED-DECIMAL, or BINARY.

Depending on the keywords specified for the data item in DDS, the keyed sequence access path can be by algebraic value. See the ABSVAL, DIGIT, SIGNED, and ZONE keywords in the *DDS Reference*. If one of these keywords is specified, its name appears in a comment table in the COBOL source listing under the heading TYPE. If no keyword is specified, the table entry is the data type specified in DDS. The table entry AN indicates that the data type is alphanumeric (specified in DDS as A). The table entry N indicates that the data type is numeric (specified in DDS as P, S, or B).

The keywords specified for the data item in DDS can modify record sequence. See the ALTSEQ, DIGIT, and ZONE keywords in the *DDS Reference*. If none of these keywords are specified, the records are ordered according to the EBCDIC collating sequence.

End of IBM Extension

The keys are ordered within the collating sequence used when the file was created.

The data description of data-name-2 and its relative location within the record must be the same as the ones used when the file was defined in DDS.

The record description that defines data-name-2 will always be used to access the record key field for the I-O operation.

IBM Extension

The reserved word EXTERNALLY-DESCRIBED-KEY can specify that the keys for this file are those that are externally described in DDS. The keys are determined by the record formats that are copied by the COPY statement, DDS, DD, DDSR, or DDR format, under the FD for this file.

The key can start at different offsets within the buffer for each format. In this situation, care must be used when changing from one record format to

another, using a random READ or START statement. The key must be placed in the record format at the correct offset in the format that will be used in the random access of the file. Unpredictable results can occur if the key for the desired record is based on data that was part of the last record read. This is because the movement of the data to the key field can involve overlapping fields.

The key within a format can be made up of multiple, noncontiguous (not adjacent) fields. Only those record formats copied in within the FD for the file should be referenced by the FORMAT phrase. If a format is referenced that is defined within the file, but that format has not been copied into the program, the key is built using the key fields defined for the first record format that was copied. This can cause unpredictable results.

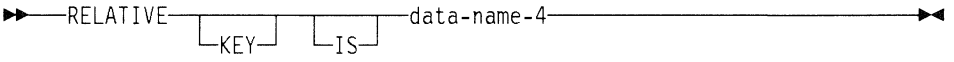
If a portion of the key is declared in the logical file only as an element of a concatenated item (rather than an independently-declared item), the result of the CONCAT operation must not be a variable-length item.

End of IBM Extension

RELATIVE KEY Clause

The RELATIVE KEY clause (Format 3) identifies a data-name that specifies the relative record number for a specific logical record within a relative file.

Format



data-name-4

Must be defined as an unsigned integer data item whose description does not contain the PICTURE symbol P. Data-name-4 must not be defined in a record description entry associated with this relative file. That is, the RELATIVE KEY is **not** part of the record.

Data-name-4 is required for ACCESS IS SEQUENTIAL only when the START statement is to be used. It is always required for ACCESS IS RANDOM and ACCESS IS DYNAMIC. When the START statement is issued, the system uses the contents of the RELATIVE KEY data item to determine the record at which sequential processing is to begin.

If a value is placed in data-name-4, and a START statement is not issued, the value is ignored and processing begins with the first record in the file.

IBM Extension

When the file is opened, the POSITION parameter on the OVRDBF CL command can be used to set the file position indicator. This causes processing to begin with a record other than the first record. See the *CL Reference* for further information.

End of IBM Extension

If a relative file is to be referenced by a START statement, you must specify the RELATIVE KEY clause for that file.

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.

Refer to the Transaction File chapter in the *COBOL/400 User's Guide* for transaction file considerations.

FILE STATUS Clause

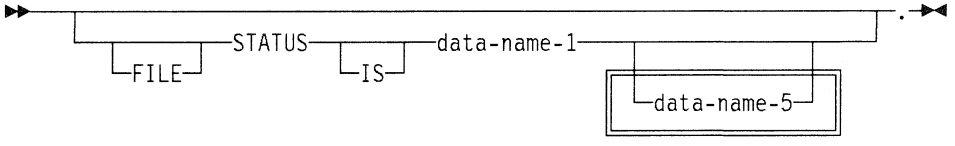
The FILE STATUS clause monitors the execution of each input-output request for the file.

When the FILE STATUS clause is specified, the system moves a value into the status key data item after each input-output request that explicitly or implicitly refers to this file. The value indicates the status of execution of the statement. (See the "Status Key" description under "Common Processing Facilities" on page 214.)

If you specify this clause at all, specify it for *all* files in a given program. For more information, see the chapter on exception and error handling in the *COBOL/400 User's Guide*.

When the compiler generates code to block output records or unblock input records, file status values that are caused by OS/400 exceptions are set only when a block is processed. See Appendix F, "File Structure Support Summary and Status Key Values" for a description of the possible values. See the *COBOL/400 User's Guide* for more information on blocking output records and unblocking input records.

Format



data-name-1

The status key data item must be defined in the Data Division as a 2-character alphanumeric item. Data-name-1 must not be defined in the File Section.

IBM Extension

data-name-1

The status key data item may also be defined in the Data Division as a 2-character numeric item, that is an unsigned integer with explicit or implicit USAGE IS DISPLAY. It is treated as an alphanumeric item.

data-name-5

An optional status key data item may be specified for transaction and non-transaction file processing. See the *COBOL/400 User's Guide* for more information about using transaction files.

For non-transaction files, the data item must be a 6-byte group item. The item is treated as documentation for all non-transaction files except for those that are dynamically created.

For transaction files, the data item must be a 4-character alphanumeric item.

End of IBM Extension

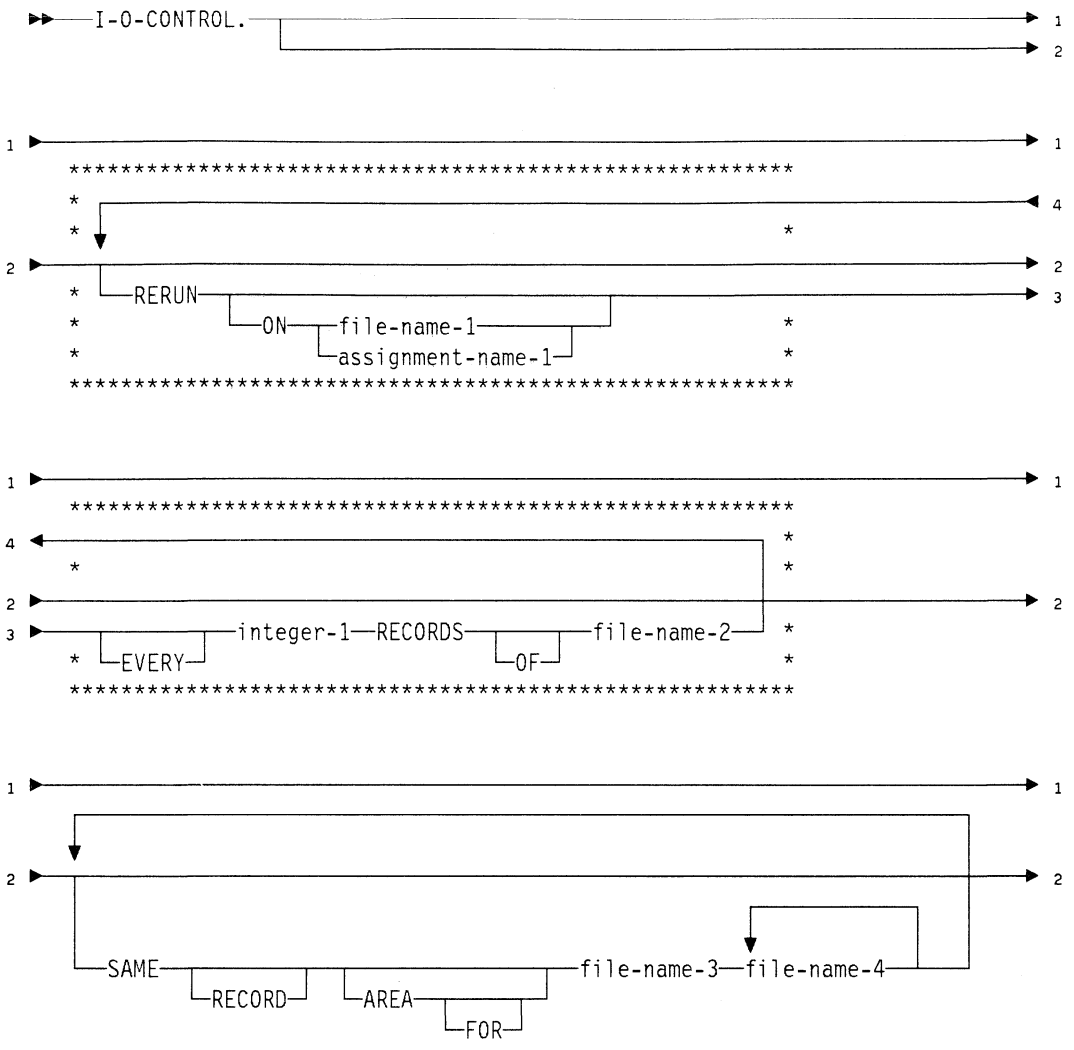
I-O-CONTROL Paragraph

The I-O-CONTROL paragraph of the INPUT-OUTPUT SECTION specifies when checkpoints are to be taken and the storage areas to be shared by different files. This paragraph is optional in a COBOL program.

The key word I-O-CONTROL may appear only once, at the beginning of the paragraph. The word I-O-CONTROL must begin in Area A, and must be followed by a separator period.

Each clause within the paragraph may be separated from the next by a separator comma or a separator semicolon. The order in which I-O-CONTROL paragraph clauses are written is not significant. The I-O-CONTROL paragraph ends with a separator period.

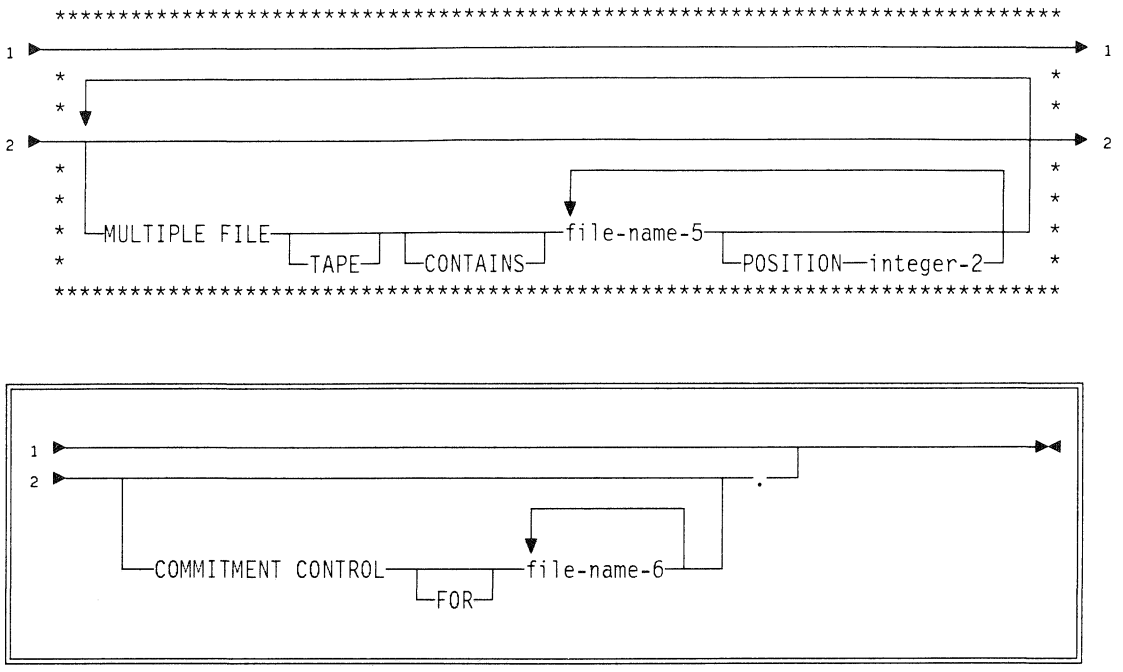
Format 1-Sequential Files



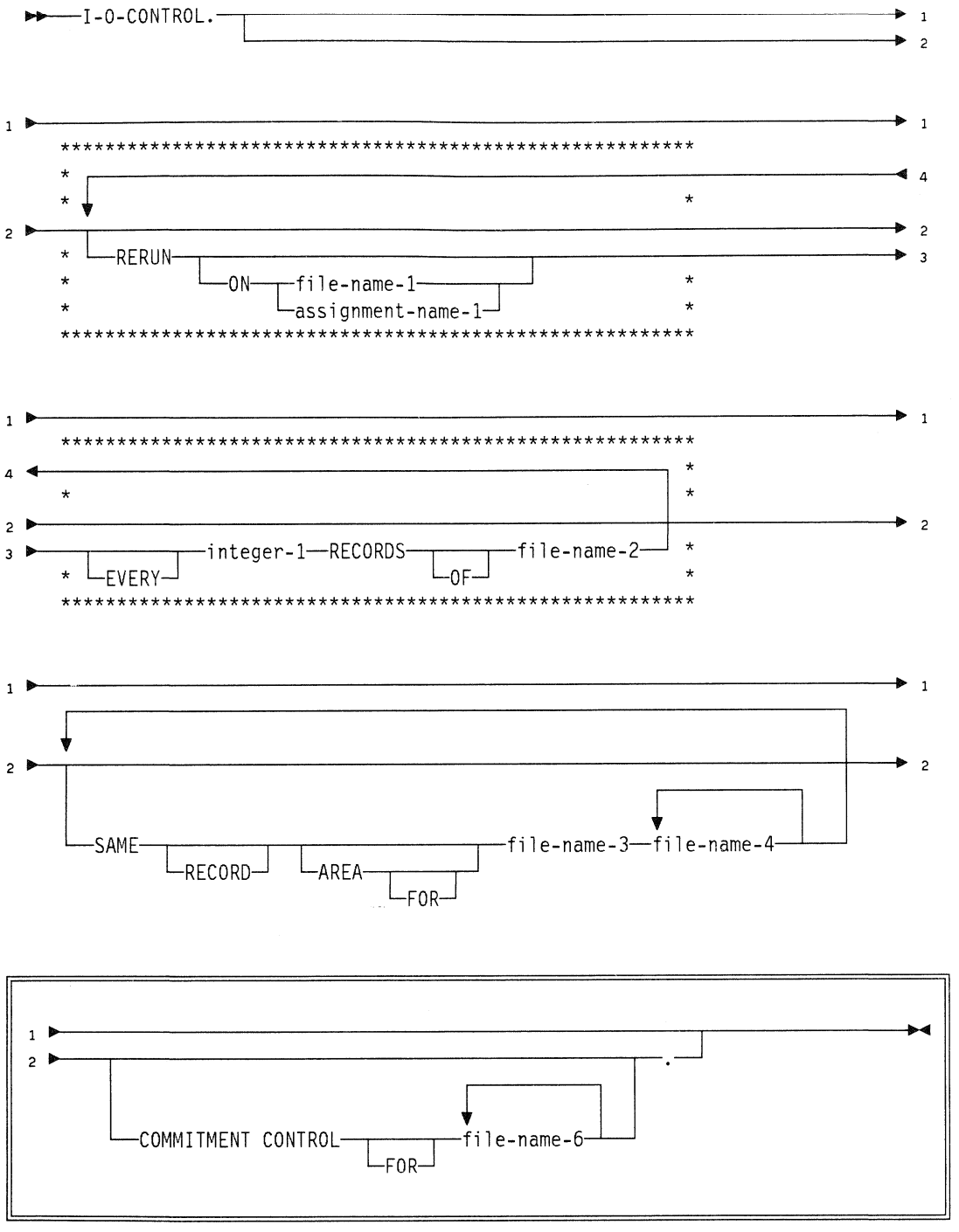
(continued on next page)

Format 1-Sequential Files

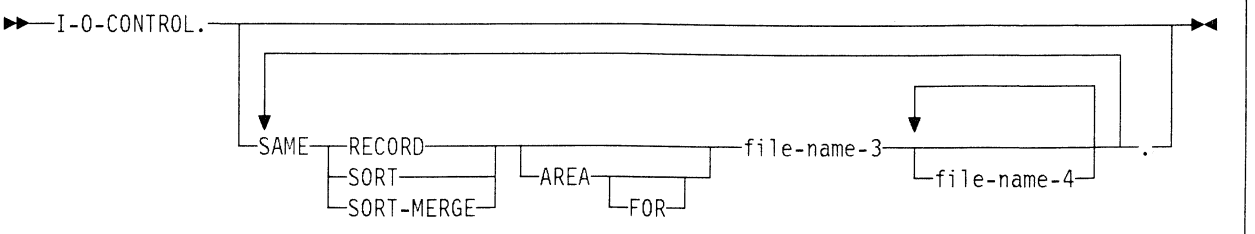
(continued from previous page)



Format 2-Indexed and Relative Files



Format 3-Sort/Merge Files

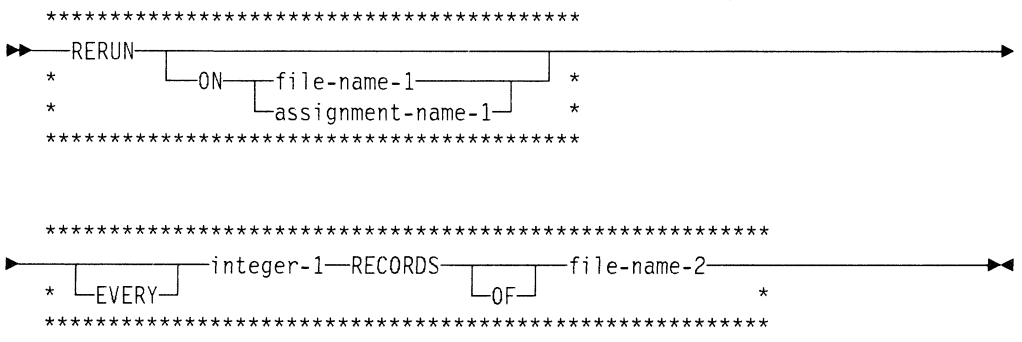


RERUN Clause

The RERUN clause specifies that checkpoint records are to be taken. Subject to the restrictions given with each phrase, more than one RERUN clause may be specified.

The RERUN clause is an obsolete element and is to be deleted from the next revision of the ANS Standard.

Format



assignment-name-1
This name can be any user defined word.

EVERY integer-1 RECORDS
A checkpoint record is to be written for every integer-1 record in file-name-2 that is processed.

When multiple integer-1 RECORDS phrases are specified, no two of them may specify the same file-name-2.

The RERUN clause is syntax-checked, but is treated as documentation.

SAME AREA Clause

The SAME AREA clause specifies that two or more files, that do not represent sort or merge files, are to use the same main storage area during processing.

The files named in a SAME AREA clause need not have the same organization or access.

The SAME AREA clause is syntax checked, but is treated as documentation. However the following should be considered for SAA compatibility:

- A specific file-name must not appear in more than one SAME AREA clause.
- If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in that SAME RECORD AREA clause. However, that SAME RECORD AREA clause can contain additional file-names that do not appear in that SAME AREA clause.
- Only one of the files for which the SAME AREA clause is specified can be open at one time. This rule takes precedence over the SAME RECORD AREA rule that all the files can be open at the same time.

SAME RECORD AREA Clause

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

- A logical record of each opened output file in the SAME RECORD AREA clause
- A logical record of the most recently read input file in the SAME RECORD AREA clause.

The SAME RECORD AREA clause allows transfer of data from one file to another with no explicit data manipulation because the input/output record areas of named files are identical, and all are available to the user.

Note: The SAME RECORD AREA clause is intended to make efficient use of main storage. However, AS/400 virtual storage architecture eliminates the need for this clause, and the clause is supported for compatibility rather than for performance. Use of the SAME RECORD AREA actually degrades performance.

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

- A specific file-name must not appear in more than one SAME RECORD AREA clause.
- If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all the file-names in that SAME AREA clause must appear in that SAME RECORD AREA clause. However, the SAME RECORD AREA clause may contain additional file-names that do not appear in the SAME AREA clause.
- The rule that in the SAME AREA clause only one file may be open at one time takes precedence over the SAME RECORD AREA rule that all the files may be open at the same time. This is for SAA compatibility purposes.

SAME SORT AREA Clause

The SAME SORT AREA clause optimizes the storage area assignment to a given SORT statement.

When the SAME SORT AREA clause is specified, at least one file-name specified must name a sort file. Files that are not sort files may also be specified. The following rules apply:

- More than one SAME SORT AREA clause may be specified. However, a given sort file must not be named in more than one such clause.
- If a file that is not a sort file is named in both a SAME AREA clause and in one or more SAME SORT AREA clauses, all the files in the SAME AREA clause must also appear in that SAME SORT AREA clause.
- Files named in a SAME SORT AREA clause need not have the same organization or access.
- Files named in a SAME SORT AREA clause that are not sort files do not share storage with each other unless the user names them in a SAME AREA or SAME RECORD AREA clause.

The SAME SORT AREA clause is syntax checked, but is treated as documentation.

SAME SORT-MERGE AREA Clause

The SAME SORT-MERGE AREA clause optimizes the storage area assignment to a given SORT or MERGE statement.

When the SAME SORT-MERGE AREA clause is specified, at least one file-name specified must name a sort or merge file. Files that are not sort or merge files may also be specified. The following rules apply:

- More than one SAME SORT-MERGE AREA clause may be specified. However, a given sort or merge file must not be named in more than one such clause.
- If a file that is not a sort or merge file is named in both a SAME AREA clause and in one or more SAME SORT-MERGE AREA clauses, all the files in the SAME AREA clause must also appear in that SAME SORT-MERGE AREA clause.
- Files named in a SAME SORT-MERGE AREA clause need not have the same organization or access.
- Files named in a SAME SORT-MERGE AREA clause that are not sort or merge files do not share storage with each other unless the user names them in a SAME AREA or SAME RECORD AREA clause.

The SAME SORT-MERGE AREA clause is syntax checked, but is treated as documentation.

MULTIPLE FILE TAPE Clause

The MULTIPLE FILE TAPE clause is syntax-checked, but is treated as documentation. This clause specifies that two or more files share the same reel of tape. The function is provided by the system through the use of command language. See CRTTAPF, CHGTAPF, and OVRTAPF commands in the *CL Reference*.

The MULTIPLE TAPE FILE clause is an obsolete element and is to be deleted from the next revision of the ANSI Standard.

IBM Extension

COMMITMENT CONTROL Clause

The COMMITMENT CONTROL clause specifies the files that will be placed under commitment control when they are opened. These files will then be affected by the COMMIT and ROLLBACK statements. The COMMIT statement allows the synchronization of changes to database records while preventing other jobs from modifying those records until the COMMIT is complete. The ROLLBACK statement provides a method of cancelling changes made to database files when those changes should not be made permanent.

The COMMITMENT CONTROL clause can specify only files assigned to a device type of DATABASE. Files under commitment control may have an organization of sequential, relative or indexed, and may have any access mode valid for a particular organization.

The system locks records contained in files under commitment control when these records are accessed. Records remain locked until released by a COMMIT or ROLLBACK statement. For more information about record locking for files under commitment control, see the *COBOL/400 User's Guide*.

Programming Note: Always try to use files in a consistent manner to avoid record locking problems, and to avoid reading records that have not yet been permanently committed to the database. Typically, a file should either always be accessed under commitment control or never be accessed under commitment control.

End of IBM Extension

Data Division

The Data Division of a COBOL source program describes, in a structured manner, all the data to be processed by the object program. The Data Division is optional in a COBOL source program.

This section outlines the structure of the Data Division and explains the types of data and how they are related in a COBOL program.

Data Division Structure

The Data Division must begin with the words DATA DIVISION, followed by a period and a space.

The Data Division is divided into three sections:

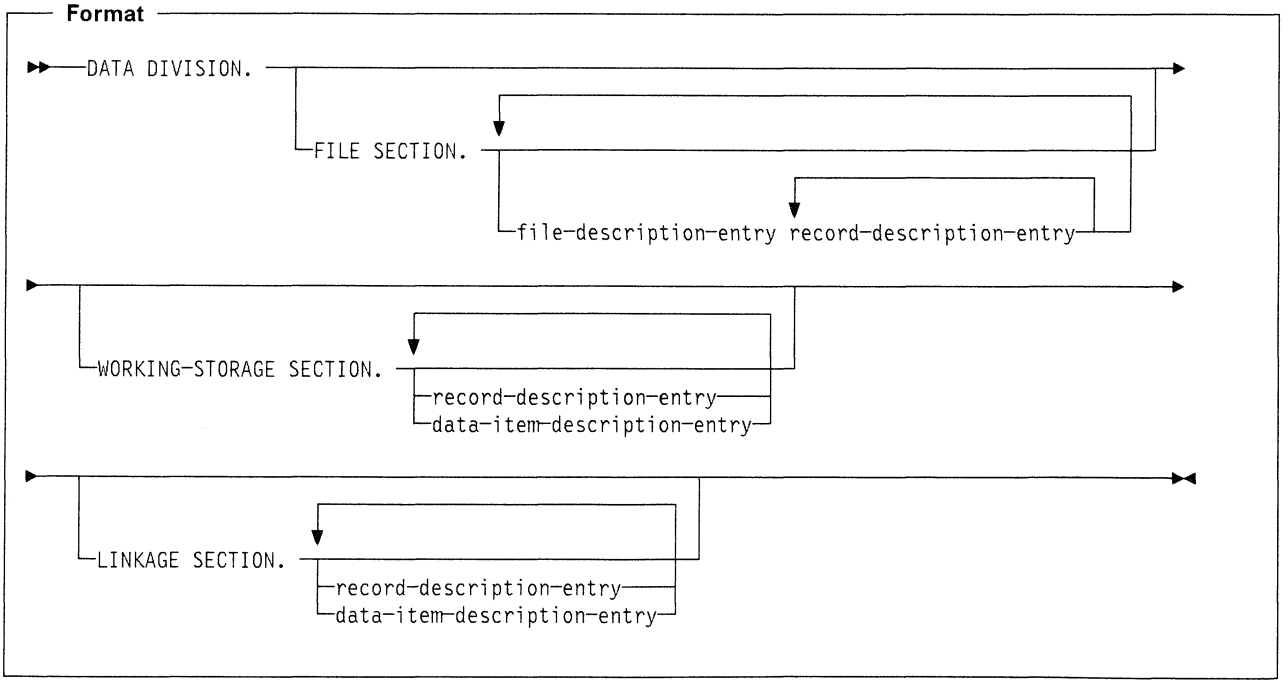
File Section
Describes externally stored data (including sort-merge files).

Working-Storage Section
Describes internal data.

Linkage Section
Describes data made available by another program.

Each section has a specific logical function within a COBOL source program, and each may be omitted from the source program when that logical function is not needed. If included, the sections must be written in the order shown.

Format



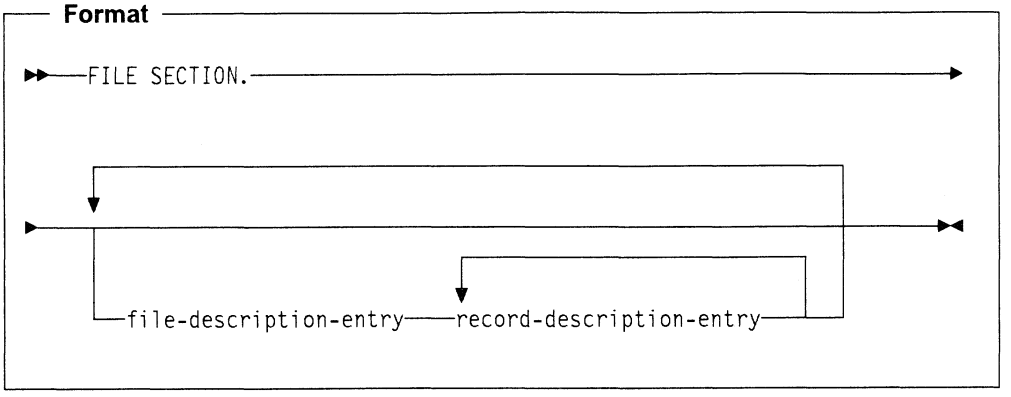
File Section

The File Section describes:

- All externally stored files
- Each sort-merge file.

The File Section must begin with the header FILE SECTION, followed by a separator period.

Format



file-description-entry
Represents the highest level of organization in the File Section. It provides information about the physical structure and identification of a file, and gives the record-name(s) associated with that file. For the format and the clauses required in a file description entry, see "Data Division—File and Sort Description Entries" on page 109.

record-description-entry
A set of data description entries (described in "Data Division—Data Description Entry" on page 125) that describe the particular record(s) contained within a particular file.

More than one record description entry may be specified; each is an alternative description of the same record storage area.

Data areas described in the File Section are not available for processing unless the file containing the data area is open.

The initial value of a data item in the File Section is undefined.

IBM Extension

The record description entry for a file can be specified using the Format 2 COPY statement (DD, DDR, DDS, or DDSR option). This allows the field descriptions for a record format to be exactly as defined in DDS. Also, programs are easier to write because the record format description is maintained in only one place. See "Part 4. Compiler-Directing Statements" on page 453 for further information on this format of the COPY statement.

End of IBM Extension

Note: Group items (including tables) are limited to a length of 32 766 bytes.

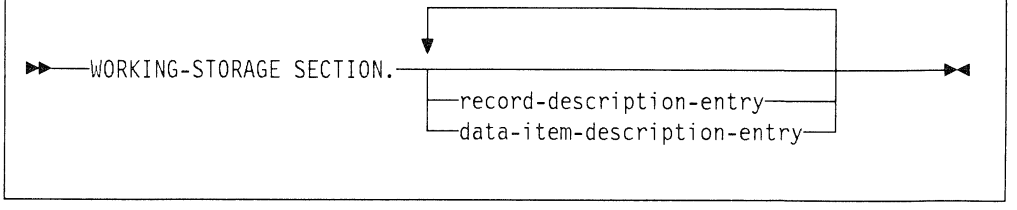
Working-Storage Section

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

The Working-Storage Section contains record description entries and data description entries for independent data items, called **data item description entries**.

The Working-Storage Section must begin with the section header WORKING-STORAGE SECTION, followed by a separator period.

Format



record-description-entry

See "File Section" on page 99 for description.

Data entries in the Working-Storage Section that bear a definite hierarchic relationship to one another must be grouped into records structured by level number.

data-item-description-entry

Independent items in the Working-Storage Section that bear no hierarchic relationship to one another need not be grouped into records, provided that they do not need to be further subdivided. Instead, they are classified and defined as independent elementary items. Each is defined in a separate data-item description entry that begins with either the level number 77 or 01. (See "Data Division—Data Description Entry" on page 125.)

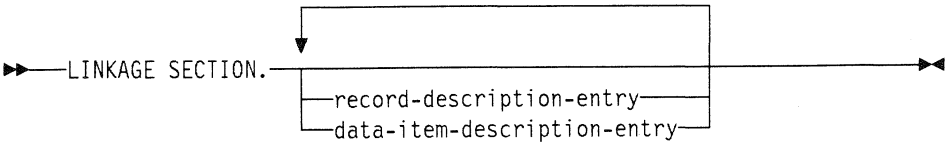
The initial value of any data item in the Working-Storage Section, except an index data item, is specified by associating a VALUE clause with the item. The initial value of any index data item, or of any data item not associated with a VALUE clause, is undefined.

Note: A maximum of 3 000 000 bytes is permitted for group items (including tables).

Linkage Section

The Linkage Section describes data made available from another program.

Format



record-description-entry
See "File Section" on page 99 for description.

data-item-description-entry
See "Working-Storage Section" on page 100 for description.

Record description entries and data item description entries in the Linkage Section provide names and descriptions, but storage within the program is not reserved because the data area exists elsewhere.

Any data description clause may be used to describe items in the Linkage Section, with one exception: the VALUE clause may not be specified for items other than level-88 items.

Note: A maximum of 3 000 000 bytes are permitted for group items (including tables).

IBM Extension

ADDRESS OF

ADDRESS OF refers to the calculated address of a data item. The data item can be reference modified or subscripted.

The ADDRESS OF any Linkage Section item except for level-number 66 or 88 can be taken. Such an address can be referenced, but not changed.

The ADDRESS OF an item is implicitly defined as USAGE IS POINTER.

ADDRESS OF Special Register

The ADDRESS OF special register is the starting address of the data structure from which all calculated addresses are determined.

It exists for each record (level-number 01 or 77) in the Linkage Section, except for those records that redefine each other. In such cases, the special register is similarly redefined.

This special register is implicitly defined as USAGE IS POINTER, and you can change it.

If you reference modify this data item, it is no longer the starting address of a data structure. It is a calculated address, or ADDRESS OF.

End of IBM Extension

Data Types

Two types of data can be processed: file data and program data.

File Data (External Data)

File data is contained in files. (See "File Section" on page 115.) A **file** is a collection of data records existing on some input-output device. A file can be considered as a group of physical records; it can also be considered as a group of logical records. The Data Division describes the relationship between physical and logical records.

A **physical record** is a unit of data that is treated as an entity when moved into or out of storage. The size of a physical record is determined by the particular input-output device on which it is stored. The size does not necessarily have a direct relationship to the size or content of the logical information contained in the file.

A **logical record** is a unit of data whose subdivisions have a logical relationship. A logical record may itself be a physical record (that is, be contained completely within one physical unit of data); several logical records may be contained within one physical record, or one logical record may extend across several physical records.

File description entries specify the physical aspects of the data (such as the size relationship between physical and logical records, the size and name(s) of the logical record(s), labeling information, and so forth).

Record description entries describe the logical records in the file, including the category and format of data within each field of the logical record, different values the data might be assigned, and so forth.

After the relationship between physical and logical records has been established, only logical records are made available to you. For this reason, a reference in this manual to "records" means logical records, unless the term "physical records" is used.

Program Data (Internal Data)

Program data is created by the program itself, instead of being read from a file.

The concept of logical records applies to program data as well as to file data. Program data can thus be grouped into logical records, and be defined by a series of record description entries. Items that need not be so grouped can be defined in independent data description entries (called **data item description entries**).

Data Relationships

The relationships among all data to be used in a program are defined in the Data Division, through a system of level indicators and level-numbers.

A **level indicator**, with its descriptive entry, identifies each file in a program. Level indicators represent the highest level of any data hierarchy with which they are associated; FD is the file description level indicator and SD is the sort-merge file description level indicator.

A **level-number**, with its descriptive entry, indicates the properties of specific data. Level-numbers can be used to describe a data hierarchy; they can indicate that this data has a special purpose, and while they can be associated with (and subordinate to) level indicators, they can also be used independently to describe internal data or data common to two or more programs. (See "Level-Numbers" on page 128 for level-number rules.)

Levels of Data

After a record has been defined, it can be subdivided to provide more detailed data references.

For example, in a customer file for a department store, one complete record could contain all data pertaining to one customer. Subdivisions within that record could be: customer name, customer address, account number, department number of sale, unit amount of sale, dollar amount of sale, previous balance, plus other pertinent information.

The basic subdivisions of a record (that is, those fields not further subdivided) are called **elementary items**. Thus, a record can be made up of a series of elementary items, or it may itself be an elementary item.

It may be necessary to refer to a set of elementary items; thus, elementary items can be combined into **group items**. Groups themselves can be combined into a more inclusive group that contains one or more subgroups. Thus, within one hierarchy of data items, an elementary item can belong to more than one group item.

A system of level-numbers specifies the organization of elementary and group items into records. Special level-numbers are also used; they identify data items used for special purposes.

Levels of Data in a Record Description Entry

Each group and elementary item in a record requires a separate entry, and each must be assigned a level-number.

A level-number is a 1- or 2-digit integer between 01 and 49, or one of three special level-numbers: 66, 77, or 88. The following level-numbers are used to structure records:

01 This level-number specifies the record itself, and is the most inclusive level-number possible. A level-01 entry may be either a group item or an elementary item. It must begin in Area A.

02-49
: These level-numbers specify group and elementary items within a record. They may begin in Area A or Area B. Less inclusive data items are assigned higher (not necessarily consecutive) level-numbers in this series.

A group item includes all group and elementary items following it, until a level-number less than or equal to the level-number of this group is encountered.

All elementary or group items immediately subordinate to one group item must be assigned identical level-numbers higher than the level-number of this group item.

IBM Extension

The COBOL/400 compiler accepts nonstandard level-numbers that are not identical to others at the same level. For example, the following two data description entries are equivalent:

```
01   EMPLOYEE-RECORD.
     05  EMPLOYEE-NAME.
         10  FIRST     PICTURE  X(10).
         10  LAST      PICTURE  X(10).
     05  EMPLOYEE-ADDRESS.
         10  STREET    PICTURE  X(10).
         10  CITY      PICTURE  X(10).
01   EMPLOYEE-RECORD.
     05  EMPLOYEE-NAME.
         10  FIRST     PICTURE  X(10).
         10  LAST      PICTURE  X(10).
     04  EMPLOYEE-ADDRESS.
         08  STREET    PICTURE  X(10).
         08  CITY      PICTURE  X(10).
```

IBM does not recommend such coding practices, and this extension is provided only for compatibility.

End of IBM Extension

Figure 5 on page 105 illustrates the concept. Note that all groups immediately subordinate to the level-01 entry have the same level-number. Note also that elementary items from different subgroups do not necessarily have the same level numbers, and that elementary items can be specified at any level within the hierarchy.

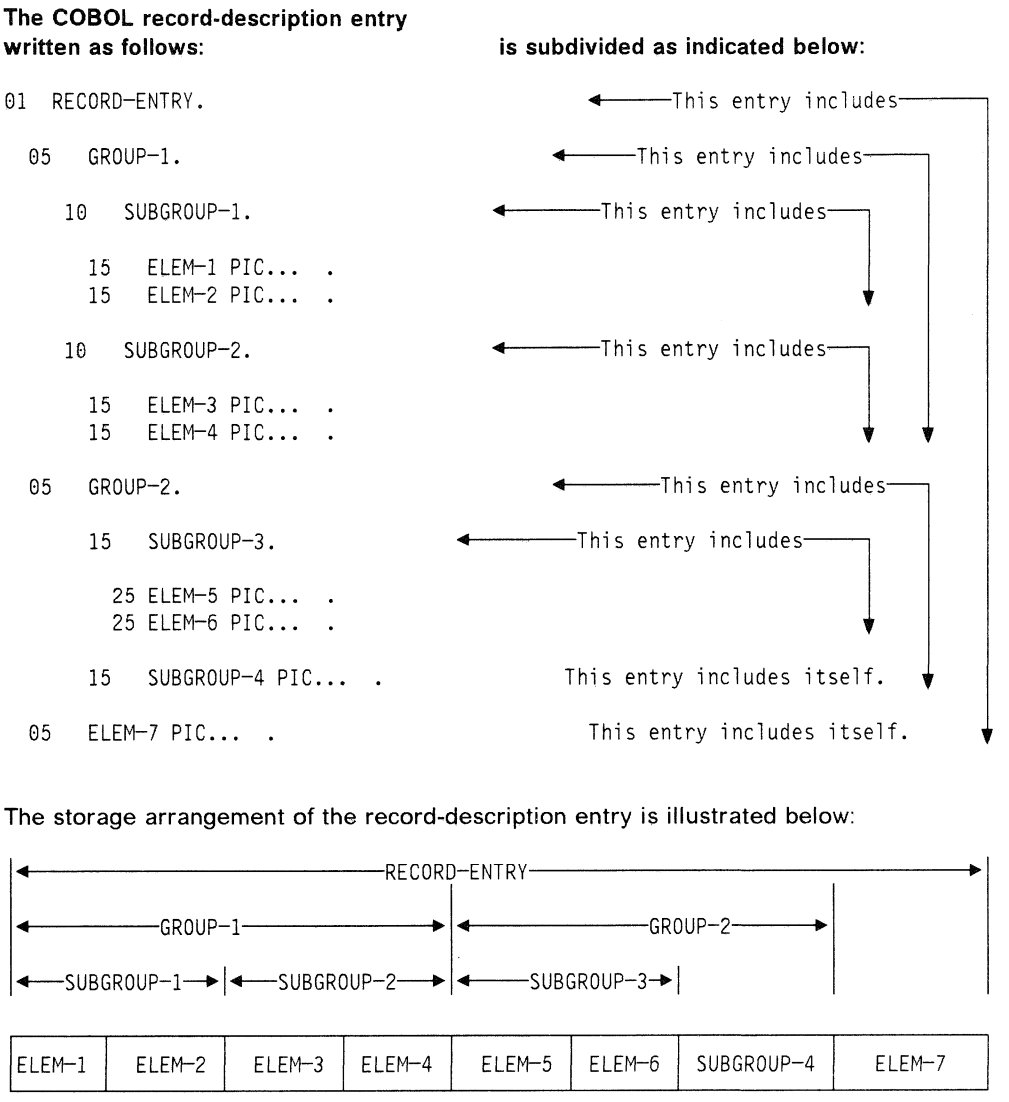


Figure 5. Levels in a Record Description

Special Level-Numbers

Special level-numbers identify items that do not structure a record. The special level-numbers are:

66 Identifies items that must contain a RENAMES clause; such items regroup previously defined data items.

(For details, see "RENAMES Clause" on page 162.)

77 Identifies data item description entries — independent Working-Storage or Linkage Section items that are not subdivisions of other items, and are not subdivided themselves. Level-77 items must begin in Area A.

88 Identifies any condition-name entry that is associated with a particular value of a conditional variable. (For details, see "VALUE Clause" on page 177.)

Note: Level-77 and level-01 entries in the Working-Storage and Linkage Sections that are referenced in the program must be given unique data-names, because

neither can be qualified. Subordinate data-names that are referenced in the program must be either uniquely defined, or made unique through qualification. Unreferenced data-names need not be uniquely defined.

Indentation

Successive data description entries may begin in the same column as preceding entries, or may be indented. Indentation is useful for documentation, but does not affect the action of the compiler.

Classes and Categories of Data

All data used in a COBOL program can be divided into four classes and six categories. Every elementary item in a program belongs to one of the classes as well as one of the categories. Every group item belongs to the alphanumeric class even if the subordinate elementary items belong to another class and category. Table 7 shows the relationship of data classes and categories.

The data category of an item is determined by its PICTURE character-string and BLANK WHEN ZERO attribute. For details, see "Data Categories and PICTURE Rules" on page 151.

IBM Extension

Boolean data is an IBM extension that provides a means of modifying and passing the values of the indicators associated with the display screen formats and externally described printer files. A Boolean value of 0 is the **off** status of the indicator, and a Boolean value of 1 is the **on** status of the indicator.

A **Boolean literal** contains a single 0 or 1, is enclosed in quotation marks, and is immediately preceded by an identifying B. A Boolean literal is defined as either B"0" or B"1".

A Boolean character occupies one byte.

When the figurative constant ZERO is associated with a Boolean data item or a Boolean literal, it represents the Boolean literal B"0".

The reserved word ALL is valid with a Boolean literal.

End of IBM Extension

Table 7. Classes and Categories of Data

Level of Item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric Edited
		Alphanumeric Edited
		Alphanumeric
	IBM Extension Boolean End of IBM Extension	IBM Extension Boolean End of IBM Extension
Group	Alphanumeric	Alphabetic
		Numeric
		Numeric Edited
		Alphanumeric Edited
		Alphanumeric
		IBM Extension Boolean End of IBM Extension

Alignment Rules

The standard alignment rules for positioning data in an elementary item depend on the category of a receiving item (that is, an item into which the data is moved; see "Elementary Moves" on page 323).

Numeric: For such receiving items, the following rules apply:

1. The data is aligned on the assumed decimal point (PICTURE character V), and if necessary, truncated or padded with zeros. (An **assumed decimal point** is one that has logical meaning but that does not exist as an actual character in the data.)
2. If an assumed decimal point is not explicitly specified, the receiving item is treated as though an assumed decimal point is specified immediately to the right of the field. The data is then treated according to the preceding rule.

Numeric-edited: The data is aligned on the decimal point, and (if necessary) truncated or padded with zeros at either end, except when editing causes replacement of leading zeros.

Alphanumeric, Alphanumeric-edited, Alphabetic: For these receiving items, the following rules apply:

1. The data is aligned at the leftmost character position, and (if necessary) truncated or padded with spaces at the right.
2. If the JUSTIFIED clause is specified for this receiving item, the above rule is modified, as described in "JUSTIFIED Clause" on page 130.

Standard Data Format

COBOL makes data description as machine independent as possible. For this reason, the properties of the data are described in relation to a standard data format rather than a machine-oriented format. For the COBOL/400 language, the default data format is the EBCDIC character set.

Character-String and Item Size

In your program, the size of an elementary item is determined through the number of character positions specified in its PICTURE character-string. In storage, however, the size is determined by the actual number of bytes the item occupies, as determined by the combination of its PICTURE character-string and its USAGE clause.

Normally, when an arithmetic item is moved from a longer field into a shorter one, the compiler truncates the data to the number of characters represented in the shorter item's PICTURE character-string.

For example, if a sending field with PICTURE S99999, and containing the value +12345, is moved to a BINARY receiving field with PICTURE S99, the data is truncated to +45. For additional information see "USAGE Clause" on page 171.

Signed Data

There are two categories of algebraic signs used in COBOL: operational signs and editing signs.

Operational Signs

Operational signs (+, -) are associated with signed numeric items, and indicate their algebraic properties. The internal representation of an algebraic sign depends on the item's USAGE clause, and, optionally, upon its SIGN clause. Zero is considered a unique value, regardless of the operational sign. An unsigned field is always assumed to be either positive or zero.

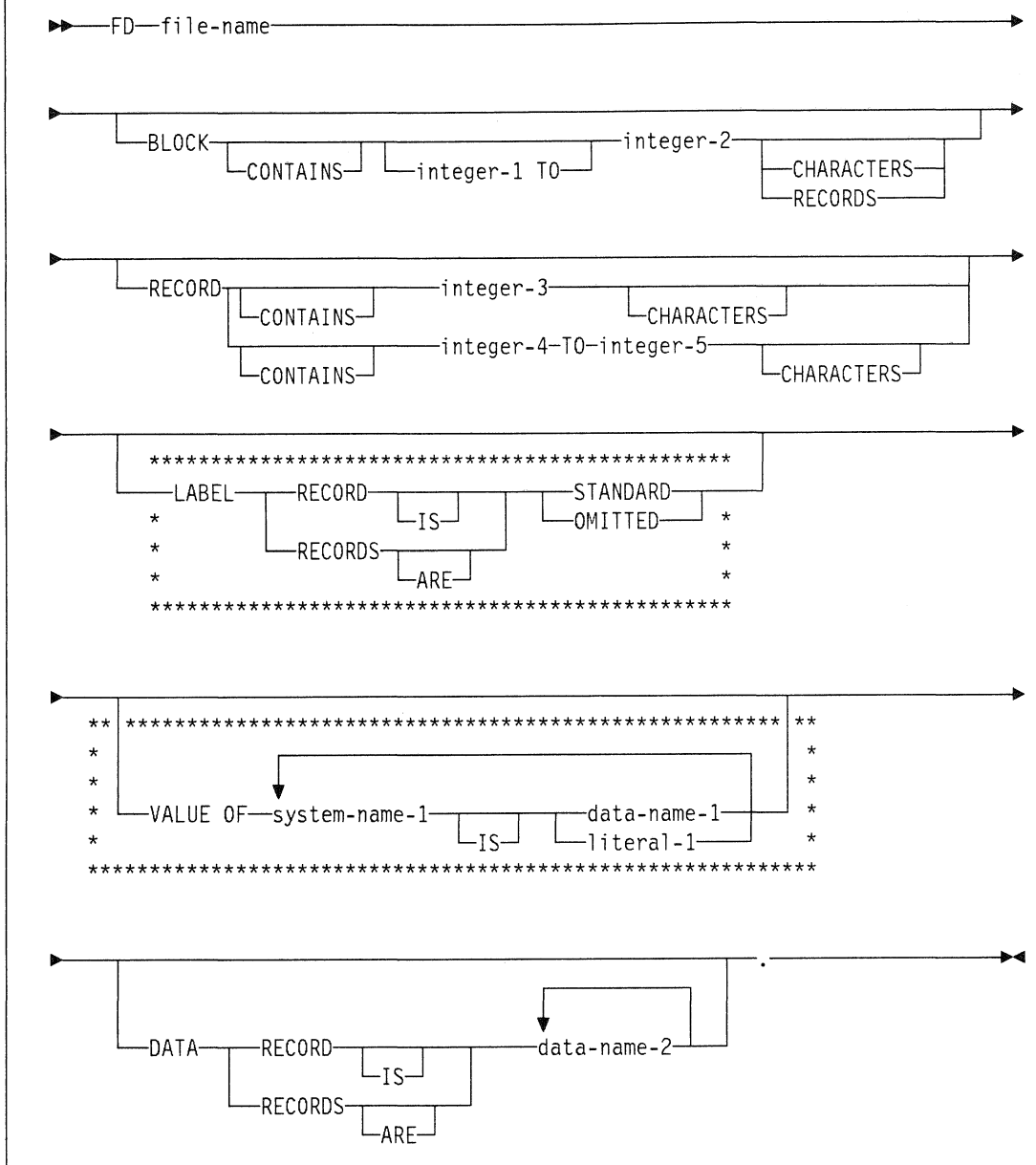
Editing Signs

Editing signs are associated with numeric edited items; editing signs are PICTURE symbols (+, -, CR, DB) that identify the sign of the item in edited output.

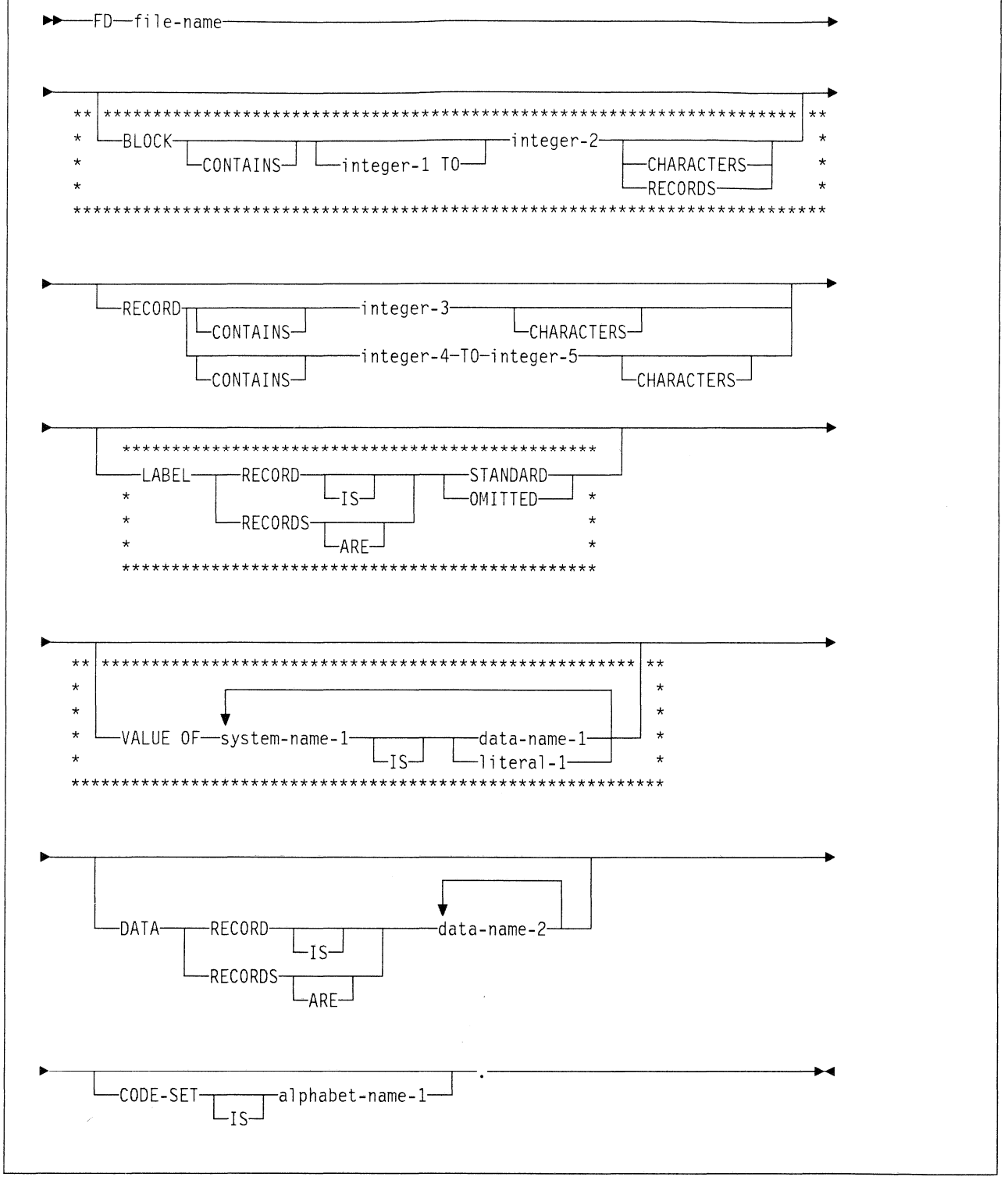
Data Division—File and Sort Description Entries

In a COBOL program, the **File Description (FD) Entry** (or **Sort Description (SD) Entry** for sort/merge files) represents the highest level of organization in the File Section.

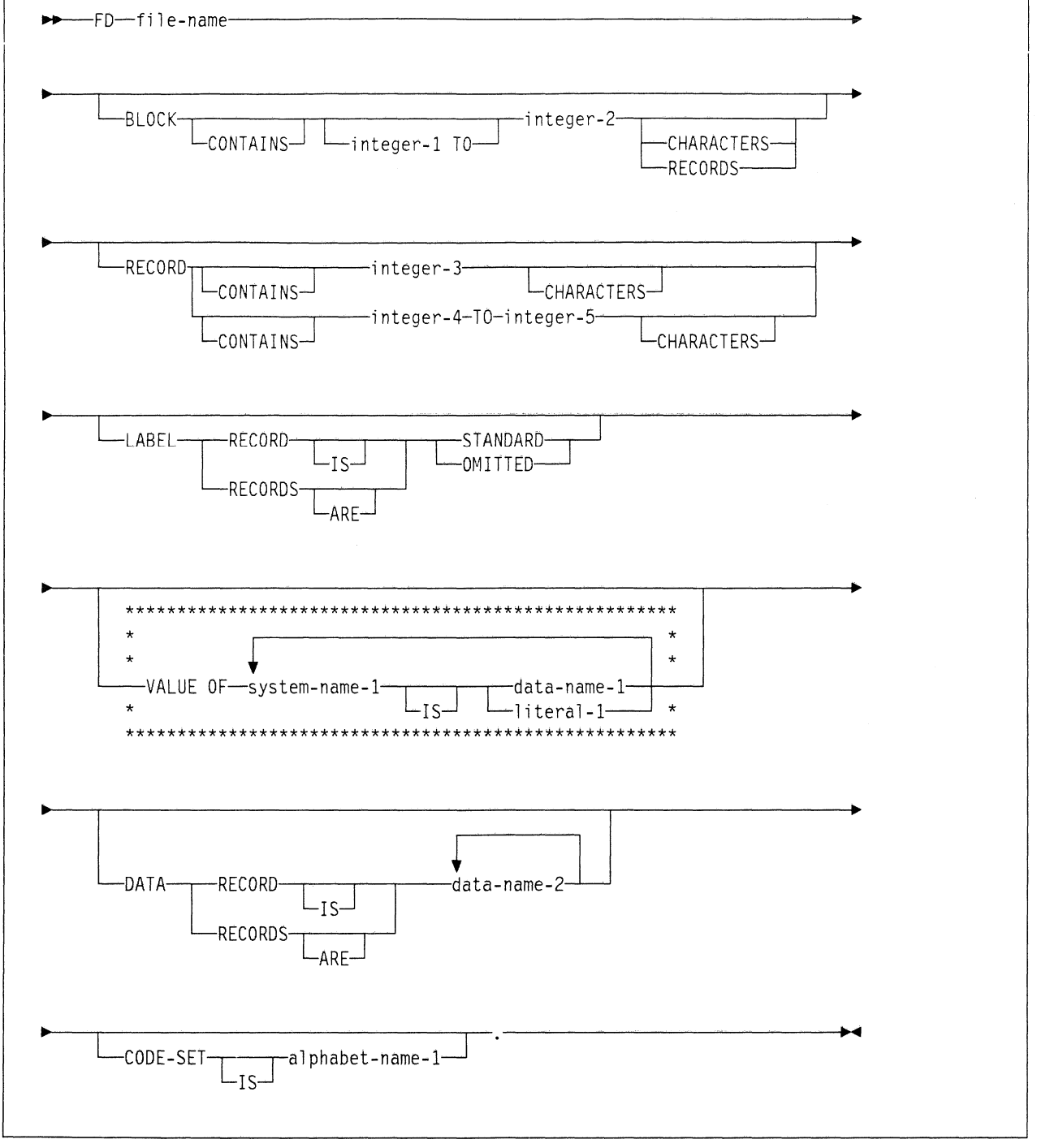
Format 1-Formatfile, Database, Disk



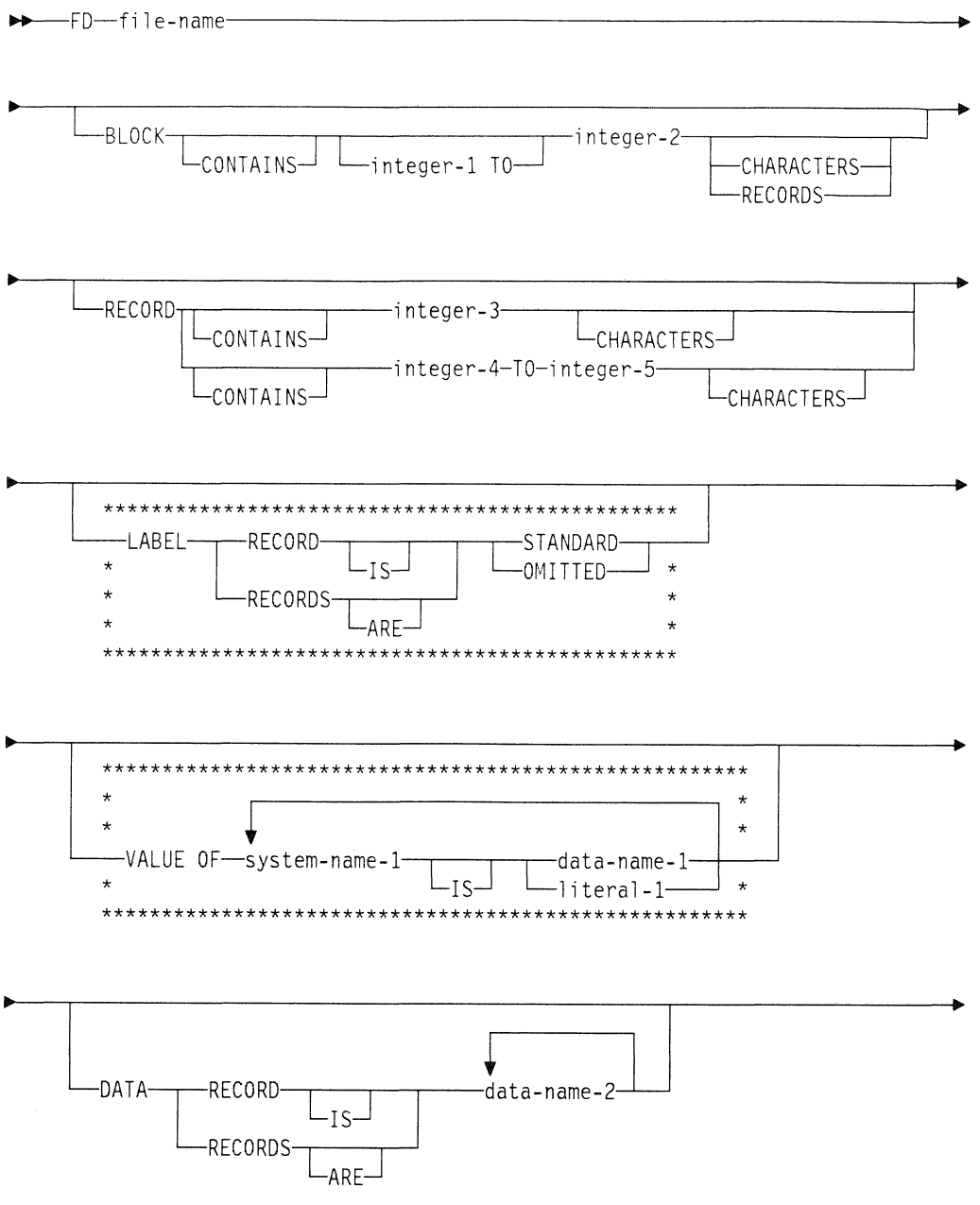
Format 2-Diskette



Format 3-Tapefile



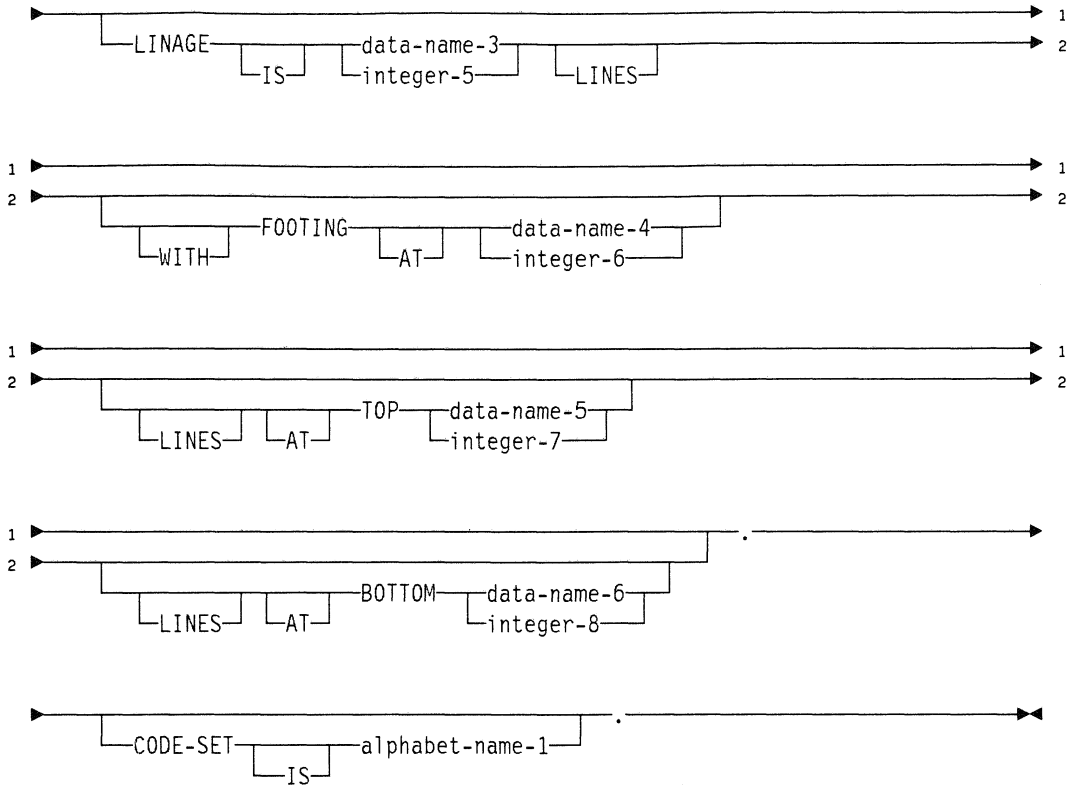
Format 4-Printer



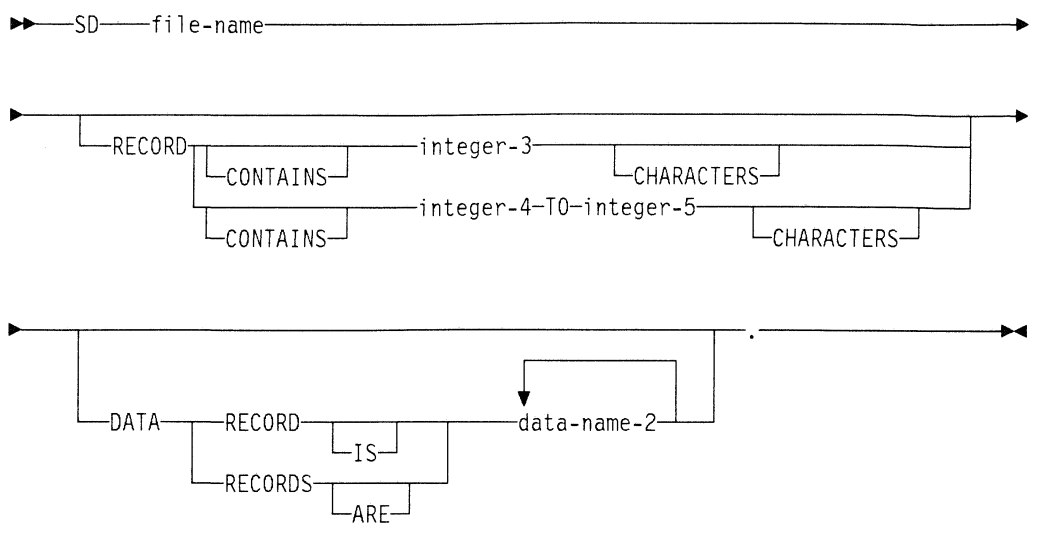
(continued on next page)

Format 4-Printer

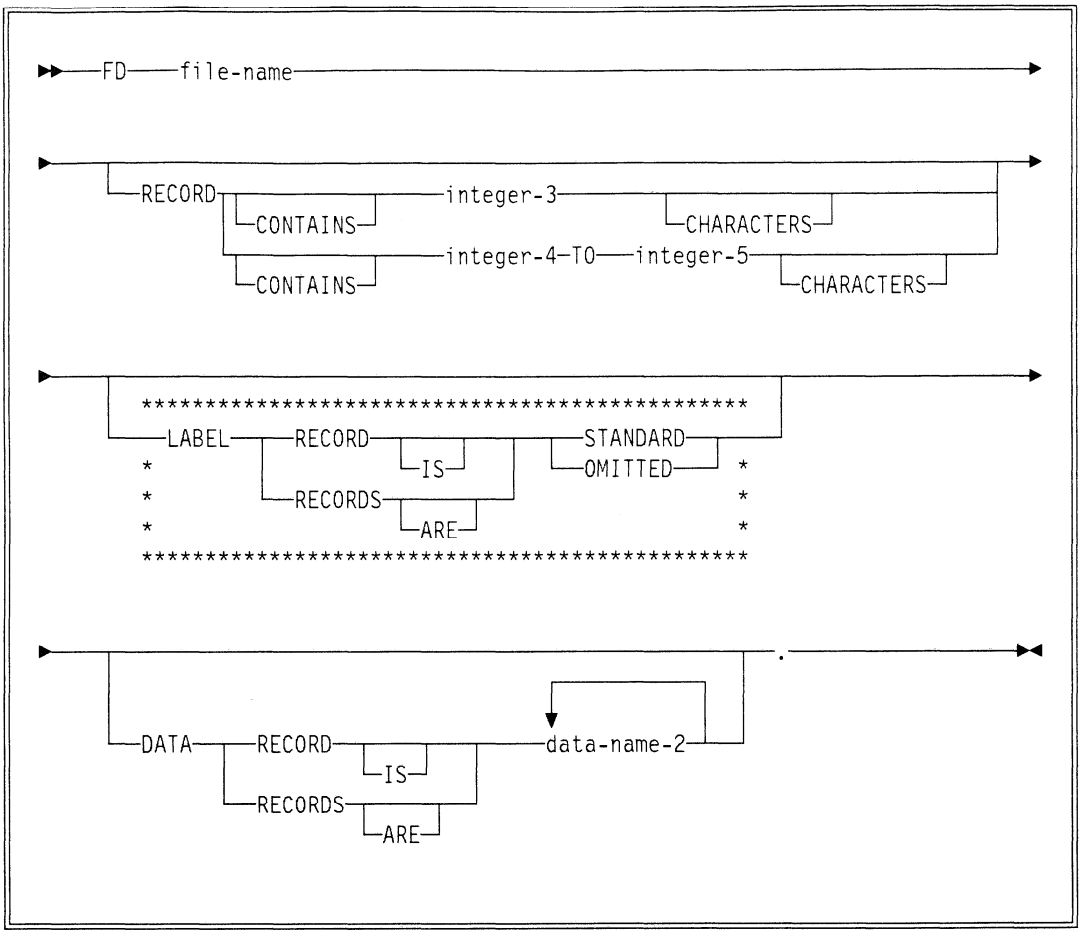
(continued from previous page)



Format 5-Sort/Merge



Format 6-Transaction



File Section

The File Section must contain a level indicator for each input and output file.

- For all files except sort/merge, the File Section must contain an FD entry.
- For each sort or merge file, the File Section must contain an SD entry.

file-name

Must follow the level indicator (FD or SD), and must be the same as that specified in the associated SELECT clause. The file-name must adhere to the rules of formation for a user-defined word; at least one character must be alphabetic. The file-name must be unique within this program.

One or more record description entries must follow the file-name. When more than one record description entry is specified, each entry implies a redefinition of the same storage area.

The clauses that follow file-name are optional; they may appear in any order.

FD (Formats 1, 2, 3, 4, and 6)
The last clause in the FD entry must be immediately followed by a separator period.

SD (Format 5)
An SD entry must be written for each sort or merge file in the program. The last clause in the SD entry must be immediately followed by a separator period.

The following example illustrates the File Section entries needed for a sort or merge file:

```
SD  SORT-FILE.

01  SORT-RECORD  PICTURE X(80).
```

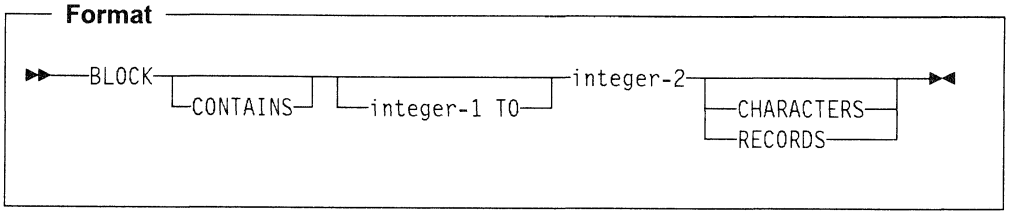
BLOCK CONTAINS Clause

The BLOCK CONTAINS clause specifies the size of the physical records.

If the records in the file are not blocked, the BLOCK CONTAINS clause may be omitted. Thus, this clause can be omitted when each physical record contains only one complete logical record.

This clause is always active for tape files, but is syntax-checked only for diskette files. To activate this clause for other types of files, use GENOPT option *BLK, or PROCESS option BLK.

Format



integer-1, integer-2
Must be nonzero unsigned integers. They specify the number of:

CHARACTERS
Specifies the number of character positions required to store the physical record, no matter what USAGE the characters have within the data record.

If only integer-2 is specified, it specifies the exact character size of the physical record. When integer-1 and integer-2 are both specified, they represent, respectively, the minimum and maximum character sizes of the physical record.

Integer-1 and integer-2 must include any control bytes and padding contained in the physical record. (Logical records do not include padding.)

For non-tape files, only integer-2 controls the blocking factor. If integer-2 is zero, the system default blocking factor applies.

The CHARACTERS phrase is the default. CHARACTERS must be specified when:

- The physical record contains padding.

Each variable record contains a four-byte header and each block contains a four-byte header when the data is transferred to tape. However, these four-byte headers are provided by the system and are of no concern to the COBOL user except that the maximum size of a variable record is restricted to 32 759.

When variable records are used, the BLOCK CONTAINS clause specifies the maximum physical record length, while the logical record length for each record is inferred by the compiler from the record name used in a WRITE statement. If an explicit length is required after a READ statement, the user can obtain it through the I-O-FEEDBACK mnemonic-name.

RECORDS
Specifies the number of logical records contained in each physical record.

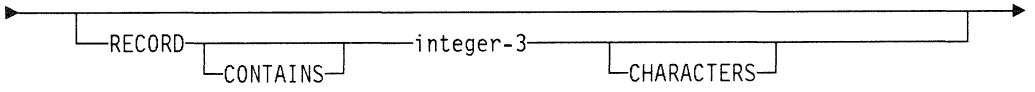
Maximum record size is 32 766; maximum block size is 32 766. These maximums include any control bytes required for variable blocked records; thus, the maximum size data record for a variable-blocked record is 32 759.

For information about record blocking, see the *COBOL/400 User's Guide*.

RECORD CONTAINS Clause

Format 1 specifies the number of character positions for fixed length records.

Format 1

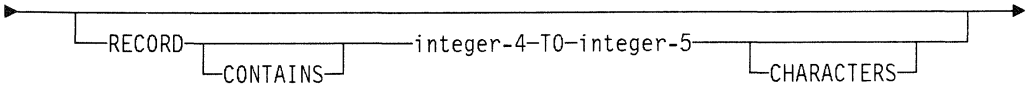


integer-3
Must be an unsigned integer that specifies the number of character positions contained in each record in the file.

When the record length determined from the record description entries does not match the length specified in the RECORD CONTAINS clause, the former will be used.

Format 2 specifies the number of character positions for either fixed or variable length records.

Format 2



integer-4, integer-5
Must be unsigned integers. Integer-4 specifies the size of the smallest data record, and integer-5 specifies the size of the largest data record.

The record size must be specified as the number of character positions needed to store the record internally. That is, it must specify the number of bytes occupied internally by the characters of the record (not the number of characters used to represent the item within the record). The size of a record is determined according to the rules for obtaining the size of a group item. (See "USAGE Clause" on page 171 and "SYNCHRONIZED Clause" on page 166.)

When one of the entries within a record description contains an OCCURS DEPENDING ON clause, the compiler calculates the record size as follows:

- When you specify the Format 2 RECORD CONTAINS clause, the record is considered to have a variable size; varying from the size obtained using the minimum size of the variable-length item, to the value obtained using the maximum size of the variable-length item
- When you do not specify the RECORD CONTAINS clause, or if the Format 1 RECORD CONTAINS clause is specified; the compiler uses the maximum value of the variable-length item to calculate the record length.

Programming Note: The system supports variable length physical records only for files on tape. For all other files, the logical records are truncated or padded to the length of the record as defined in the CRTxxxF CL command. User length in the following table is defined as the largest record associated with the given file, as specified by its record description.

Input/Output Type	User Length Less Than File Record Length	User Length Greater Than File Record Length
Input	Truncation	Pad with blanks.
Output	Pad with blanks	Truncation if old file (non-empty); for new (empty files) the larger record length is used.

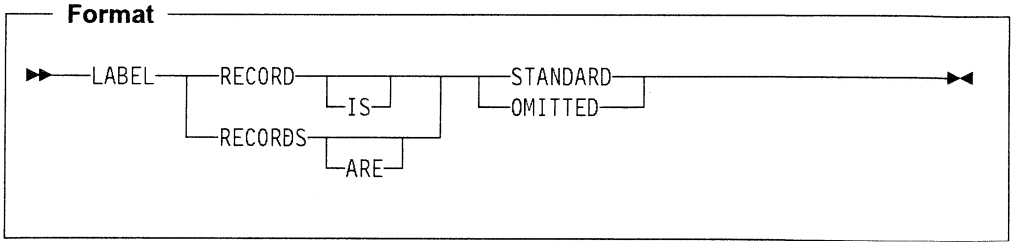
Note: The maximum record length for a file is 32 766.

LABEL RECORDS Clause

The LABEL RECORDS clause indicates the presence or absence of labels. If it is not specified for a file, label records for that file must conform to the system label specifications. FD — Format 3 (TAPEFILE) is the only format in which this clause is not treated as documentation.

Note: The LABEL RECORDS clause is an obsolete element and is to be deleted from the next revision of the ANSI Standard.

Format



STANDARD
Labels conforming to system specifications exist for this file.

OMITTED
No labels exist for this file.

IBM Extension

The LABEL RECORDS clause can be changed at execution time by specifying the REELS parameter of the Override with Tape File (OVRTAPF) CL command. See the *CL Reference* for more information on this command.

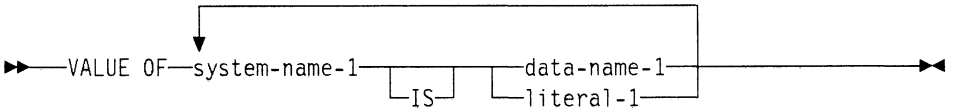
End of IBM Extension

VALUE OF Clause

The VALUE OF clause describes an item in the label records associated with this file. The clause is syntax checked, then treated as documentation.

Note: The VALUE OF clause is an obsolete element and will be deleted from the next revision of the ANSI Standard.

Format



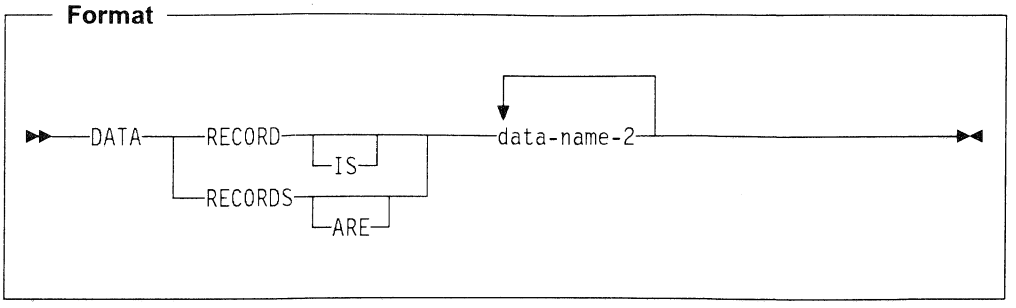
system-name-1
Must follow the rules for formation of a user-defined word.

data-name-1
Should be qualified when necessary, but cannot be subscripted. It must be described in the Working-Storage Section, and cannot be described with the USAGE IS INDEX clause.

DATA RECORDS Clause

The DATA RECORDS clause is syntax-checked, but it serves only as documentation for the names of data records associated with this file.

Note: The DATA RECORDS clause is an obsolete element and is to be deleted from the next revision of the ANSI Standard.



data-name-2
The names of record description entries associated with this file.

The specification of more than one data-name indicates that this file contains more than one type of data record. Two or more record descriptions for this file occupy the same storage area. These records need not have the same description or length. The order in which the data-names are listed is not significant.

LINAGE Clause

The LINAGE clause specifies the depth of a logical page in terms of number of lines. Optionally, it also specifies the line number at which the footing area begins, as well as the top and bottom margins of the logical page. (The logical page and the physical page may not be the same size.)

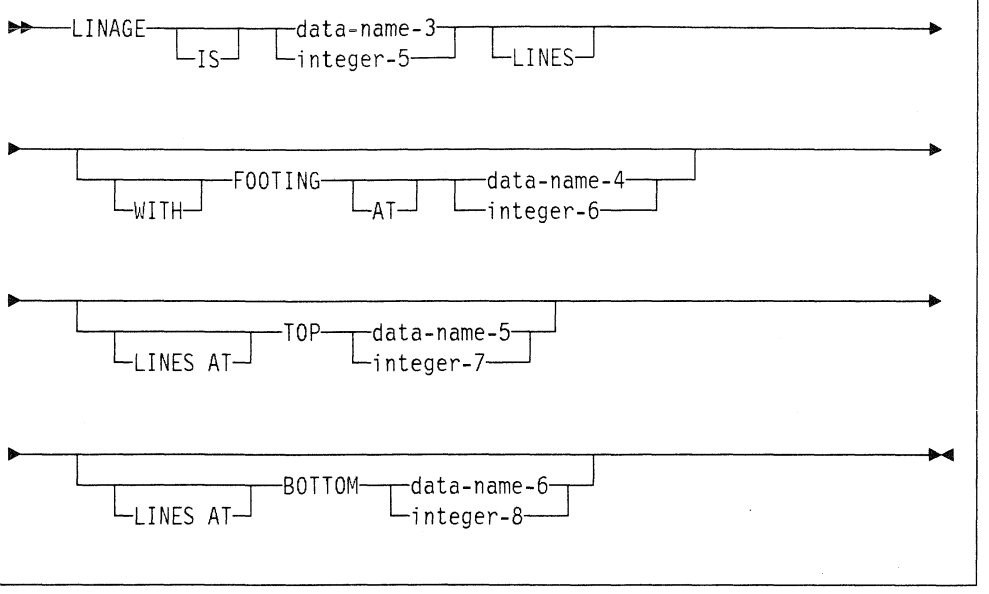
The LINAGE clause does not affect the number of lines in the selected device file; it only affects the logical page mechanism within the COBOL program.

At execution time, the printer file being used determines the physical page size. This information is used to issue appropriate space and eject commands to produce the logical page as defined in the LINAGE clause. Thus, the logical page can contain multiple physical pages, or one physical page can contain multiple logical pages.

The LINAGE clause can be specified only for files assigned to the device PRINTER. See "FILE-CONTROL Paragraph" on page 72.

All integers must be unsigned. All data-names must be described as unsigned integer data items.

Format



data-name-3, integer-5
The number of lines that can be written and/or spaced on this logical page. The area of the page that these lines represent is called the **page body**. The value must be greater than zero.

WITH FOOTING AT
Integer-6 or the value in data-name-4 specifies the first line number of the footing area within the page body. The footing line number must be greater than zero, and not greater than the last line of the page body. The footing area extends between those two lines.

LINES AT TOP
Integer-7 or the value in data-name-5 specifies the number of lines in the top margin of the logical page. The value may be zero.

LINES AT BOTTOM
Integer-8 or the value in data-name-6 specifies the number of lines in the bottom margin of the logical page. The value may be zero.

Figure 6 on page 122 illustrates the use of each phrase of the LINAGE clause.

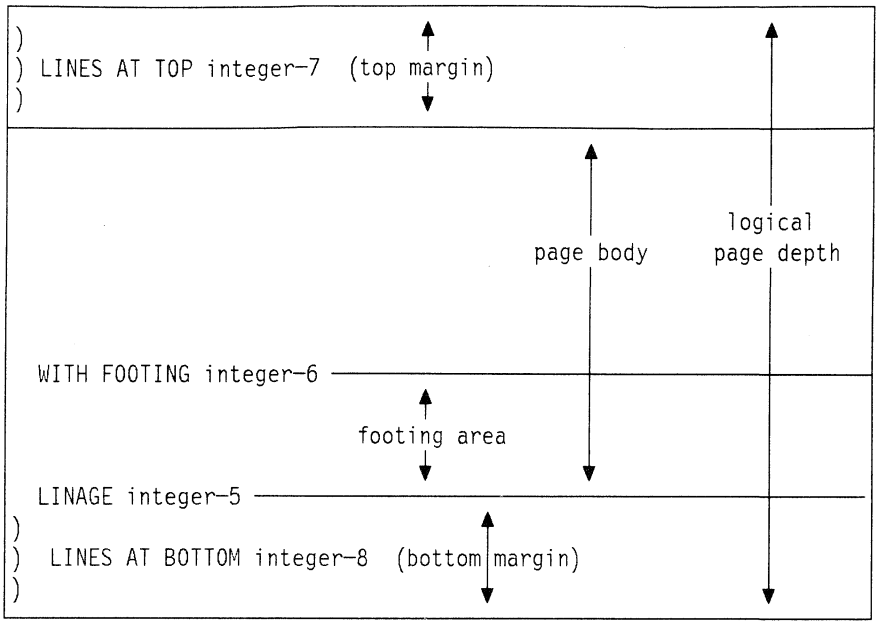


Figure 6. LINAGE Clause Phrases

The logical page size specified in the LINAGE clause is the sum of all values specified in each phrase except the FOOTING phrase. If the LINES AT TOP and/or the LINES AT BOTTOM phrase is omitted, the assumed value for top and bottom margins is zero. Each logical page immediately follows the preceding logical page, with no additional spacing provided.

If the FOOTING phrase is specified and the value of data-name-4 or integer-6 is equal to the LINAGE value of data-name-3 or integer-5, one line (the last line of the logical page) is available for footing information.

If the FOOTING phrase is not specified, no end-of-page condition independent of the page overflow condition exists.

If the FOOTING phrase is omitted, its assumed value is equal to that of the page body (integer-5 or data-name-3).

At the time an OPEN OUTPUT statement is executed, the values of integer-5, integer-6, integer-7, and integer-8, if specified, are used to determine the page body, first footing line, top margin, and bottom margin of the logical page for this file. See Figure 6. These values are then used for all logical pages printed for this file during a given execution of the program.

At the time an OPEN statement with the OUTPUT phrase is executed for the file, data-name-3, data-name-4, data-name-5, and data-name-6 determine the page body, first footing line, top margin, and bottom margin for the first logical page only.

At the time a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs, the values of data-name-3, data-name-4, data-name-5, and data-name-6, if specified, are used to determine the page body, first footing line, top margin, and bottom margin for the next logical page.

LINAGE-COUNTER Special Register

For each FD entry containing a LINAGE clause, a separate LINAGE-COUNTER special register is generated. When more than one is generated, you must qualify each LINAGE-COUNTER with its related file-name.

The implicit description of LINAGE-COUNTER is one of the following:

- If the LINAGE clause specifies data-name-3, LINAGE-COUNTER has the same PICTURE and USAGE as data-name-3.
- If the LINAGE clause specifies integer-5, LINAGE-COUNTER is a binary item large enough to hold the binary representation of integer-5.

The value in LINAGE-COUNTER at any given time is the line number at which the device is positioned within the current page. LINAGE-COUNTER may be referred to in Procedure Division statements; it cannot be modified by them.

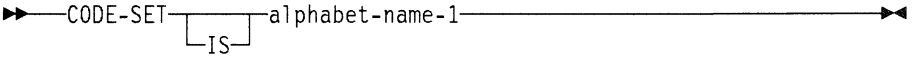
LINAGE-COUNTER is initialized to 1 when an OPEN statement for this file is executed.

LINAGE-COUNTER is automatically modified by any WRITE statement for this file. (See "WRITE Statement" on page 432.)

CODE-SET Clause

The CODE-SET clause specifies the character code used to represent data on the external media. When the CODE-SET clause is specified, an alphabet-name identifies the character code convention used to represent data on the input-output device.

Format



Alphabet-name-1 must be defined in the SPECIAL-NAMES paragraph as STANDARD-1 (for ASCII-encoded files), STANDARD-2 (for ISO 7-bit encoded files), or NATIVE (for EBCDIC-encoded files). When NATIVE is specified, the CODE-SET clause is syntax-checked, but it has no effect on the execution of the program.

The CODE-SET clause also specifies the algorithm for converting the character codes on the input-output medium from/to the internal EBCDIC character set.

When the CODE-SET clause is specified for a file, all data in this file must have USAGE DISPLAY, and, if signed numeric data is present, it must be described with the SIGN IS SEPARATE clause.

When the CODE-SET clause is omitted, the EBCDIC character set is assumed.

Note:

The CODE-SET clause may be specified for all files with sequential organization.

The AS/400 system only supports ASCII and ISO for tape and diskette files. Therefore, if the CODE-SET clause specifies a character code set of STANDARD-1 (ASCII), or STANDARD-2 (ISO) for a file that is not a tape or diskette file, a warning message is issued and the EBCDIC character set will be used.

IBM Extension

If the CODE-SET clause is omitted, the CODE parameter of the Create Diskette File (CRTDKTF) or the Create Tape File (CRTTAPF) CL command is used.

The CODE-SET clause can be changed at execution time by specifying the CODE parameter on the Override with Diskette File (OVRDKTF) or the Override with Tape File (OVRTAPF) CL command. See the *CL Reference* for more information on these commands.

End of IBM Extension

Data Division—Data Description Entry

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

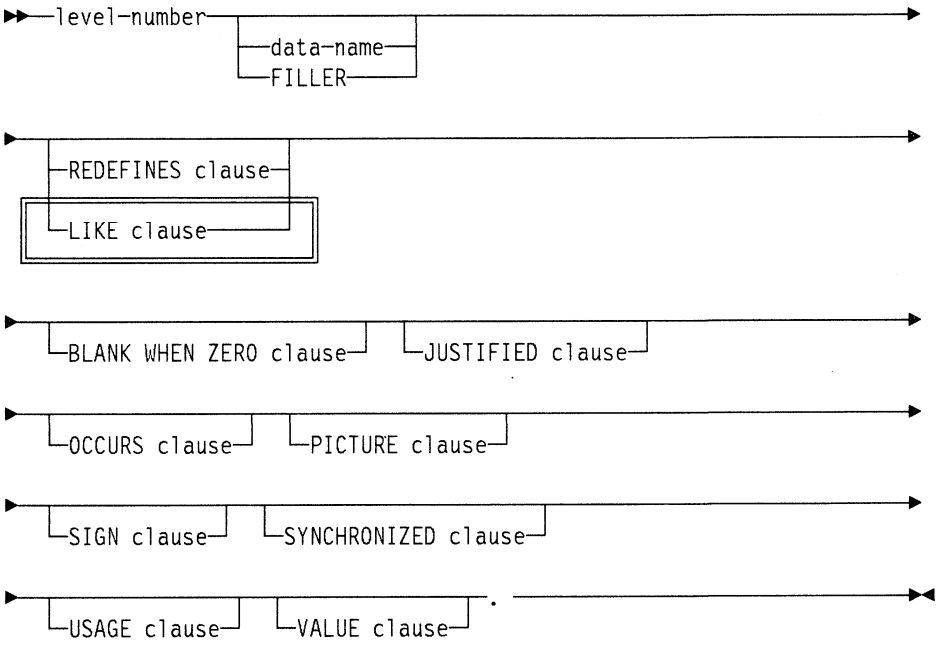
This section describes the coding of data description entries and record description entries (which are sets of data description entries). The single term **data description entry** is used in this section to refer to data and record description entries.

Data description entries that define **independent** data items do not make up a record. These are known as **data item description entries**.

The data description entry has three general formats. All data description entries must end with a separator period.

Format 1 is used for data description entries in all Data Division sections.

General Format 1



The clauses may be written in any order with two exceptions:

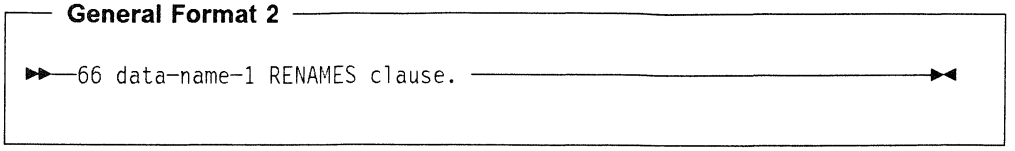
- If data-name or FILLER is specified, it must immediately follow the level-number.
- When the REDEFINES clause is specified, it must immediately follow data-name or FILLER, if either is specified. If data-name or FILLER is not specified, the REDEFINES clause must immediately follow the level-number.

Not all clauses are compatible with each other. For details, see the descriptions of the individual clauses.

A space, a separator comma, or a separator semicolon must separate clauses.

Format 2 regroups previously defined items.

General Format 2



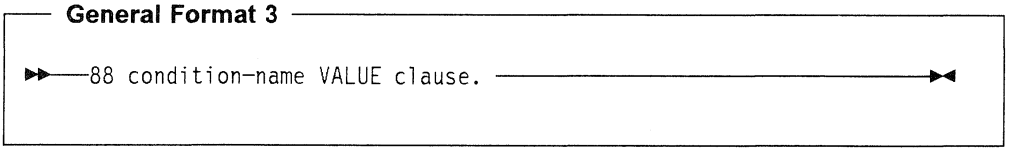
A level-66 entry cannot rename another level-66 entry, nor can it rename a level-01, level-77, or level-88 entry.

All level-66 entries associated with one record must immediately follow the last data description entry in that record.

Details are contained in "RENAMES Clause" on page 162.

Format 3 describes condition-names.

General Format 3



condition-name
A user-specified name that associates a value, a set of values, or a range of values with a conditional variable.

A **conditional variable** is a data item that can assume one or more values, that can, in turn, be associated with a condition-name.

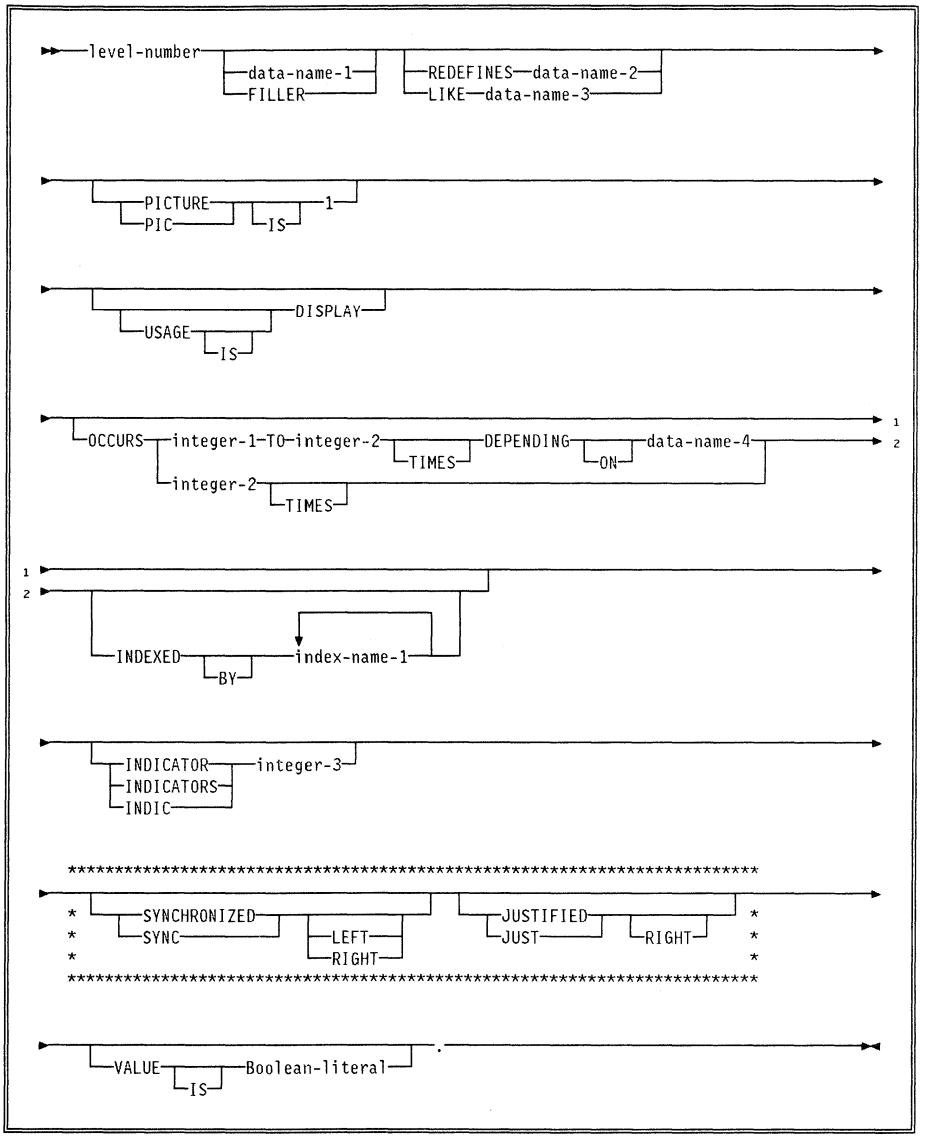
Format 3 can be used to describe both elementary and group items. Further information on condition-name entries can be found under "VALUE Clause" on page 177.

IBM Extension

Boolean data items are items that are limited to a value of 1 or 0.

Note: When you use indicators in a COBOL program, you must describe them as Boolean data items using the data description entry for Boolean data.

Format 4—Boolean Data



The special considerations for the clauses used with the Boolean data follow. All other rules for clauses are the same as those for other data.

LIKE Clause: The length of the data item cannot be changed using this clause.

PICTURE Clause: An elementary Boolean data-name is defined by a PICTURE containing a single 1.

USAGE Clause: USAGE must be defined implicitly or explicitly as DISPLAY.

OCCURS Clause: When the OCCURS clause and the INDICATOR clause are both specified at an elementary level, a table of Boolean data items is defined with each element in the table corresponding to an external indicator. The first element in the table corresponds to the indicator number specified in the INDICATOR clause; the second element corresponds to the indicator that sequentially follows the indicator specified by the INDICATOR clause.

For example, if the following is coded:

```
07   SWITCHES  PIC 1
               OCCURS 10 TIMES
               INDICATOR 16.
```

then SWITCHES (1) corresponds to indicator 16, SWITCHES (2) corresponds to indicator 17,..., and SWITCHES (10) corresponds to indicator 25.

INDICATOR Clause: If indicator fields are in a separate indicator area, the INDICATOR clause associates an indicator defined in DDS with a Boolean data item. If indicator fields are in the record area, the INDICATOR clause is syntax-checked, but is treated as documentation.

Integer-3 must be a value of 1 through 99.

The INDICATOR clause must be specified at an elementary level only. See the *COBOL/400 User's Guide* for more information on indicators.

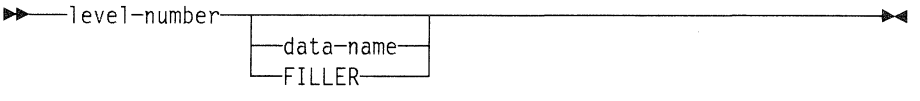
VALUE Clause: The VALUE clause specifies the initial content of a Boolean data item. The allowable values for Boolean literals are B"0", B"1", and ZERO.

End of IBM Extension

Level-Numbers

The level-number specifies the hierarchy of data within a record, and identifies special-purpose data entries. A level-number begins a data description entry, a renamed or redefined item, or a condition-name entry. A level-number has a value taken from the set of integers between 1 and 49, or from one of the special level-numbers, 66, 77, or 88.

Format



level-number

01 and 77 must begin in Area A. Level-number 01 must be followed either by a separator period; or by a space followed by its associated data-name, FILLER, or appropriate data description clause. Level-number 77 must be followed by a space followed by its associated data-name.

Level numbers 02 through 49 may begin in Area A or B and must be followed by a space or a separator period.

Level numbers 66 and 88 may begin in Area A or B and must be followed by a space.

Single-digit level-numbers 1 through 9 may be substituted for level-numbers 01 through 09.

Successive data description entries may start in the same column as the first or they may be indented according to the level-number. Indentation does not affect the magnitude of a level-number.

When level-numbers are indented, each new level-number may begin any number of spaces to the right of Area A. The extent of indentation to the right is limited only by the width of Area B.

For more information, see "Levels of Data" on page 103 and "Standard Data Format" on page 108.

IBM Extension

Elementary items or group items that are immediately subordinate to one group item can have unequal level-numbers.

End of IBM Extension

data-name

Explicitly identifies the data being described.

If specified, a data-name identifies a data item used in the program. The data-name must be the first word following the level-number.

The data item can be changed during program execution.

FILLER

Is a data item that is not explicitly referred to in a program. The key word FILLER is optional. If specified, FILLER must be the first word following the level-number.

The key word FILLER may be used with a conditional variable, if explicit reference is never made to the conditional variable but only to values it may assume. FILLER may not be used with a condition-name.

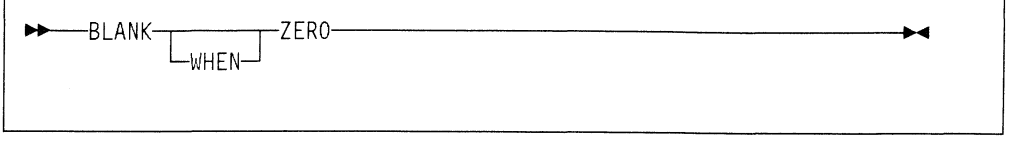
In a MOVE CORRESPONDING statement, or in an ADD CORRESPONDING or SUBTRACT CORRESPONDING statement, FILLER items are ignored.

If the data-name or FILLER clause is omitted, the data item being described is treated as though FILLER had been specified.

BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause specifies that an item contains nothing but spaces when its value is zero.

Format



The BLANK WHEN ZERO clause may be specified only for elementary numeric or numeric-edited items. These items must be described, either implicitly or explicitly, as USAGE IS DISPLAY. When the BLANK WHEN ZERO clause is specified for a numeric item, the item is considered a numeric-edited item.

The BLANK WHEN ZERO clause must not be specified for level-66 or level-88 items.

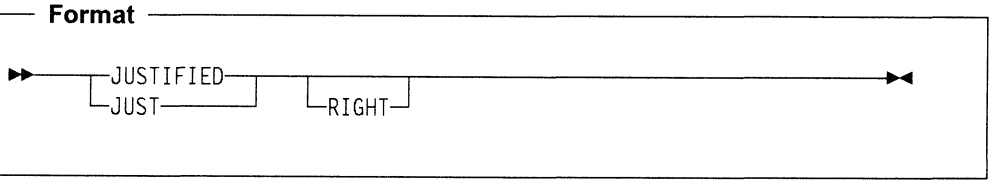
The BLANK WHEN ZERO clause must not be specified for the same entry as the PICTURE symbols S or *.

The BLANK WHEN ZERO clause is not allowed for items described with the USAGE IS INDEX clause or the USAGE IS POINTER clause.

JUSTIFIED Clause

The JUSTIFIED clause overrides standard positioning rules for a receiving item of the alphabetic or alphanumeric categories.

Format



The JUSTIFIED clause may be specified only at the elementary level. JUST is an abbreviation for JUSTIFIED, and has the same meaning. RIGHT is an optional word and has no effect on the execution of the program.

The JUSTIFIED clause cannot be specified for numeric, numeric-edited, or alphanumeric-edited items. Also, the JUSTIFIED clause cannot be specified in descriptions of items described with the USAGE IS INDEX clause or the USAGE IS POINTER clause.

IBM Extension

The JUSTIFIED clause can be specified for an alphanumeric edited item.

End of IBM Extension

When the JUSTIFIED clause is specified for a receiving item, the data is aligned at the rightmost character position in the receiving item. Also:

- If the sending item is larger than the receiving item, the leftmost characters are truncated.
- If the sending item is smaller than the receiving item, the unused character positions at the left are filled with spaces.

When the JUSTIFIED clause is omitted, the rules for standard alignment are followed (see "Alignment Rules" on page 107).

The JUSTIFIED clause must not be specified with level-66 (RENAMES) and level-88 (condition-name) entries.

The JUSTIFIED clause does not affect initial values, as determined by the VALUE clause.

The following shows the difference between standard and justified alignment when the receiving field has a length of 5 character positions:

Table 8. Justified Clause Alignment

Alignment	Sending Field Value	Receiving Field Value
Standard	THE	THE␣␣
Justified right	THE	␣␣THE
Standard	TOO␣BIG	TOO␣B
Justified right	TOO␣BIG	O␣BIG

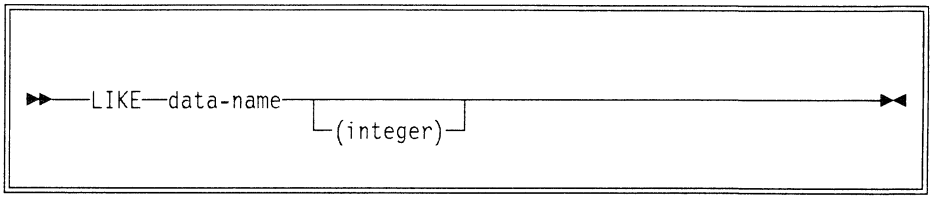
Note: You cannot use the JUSTIFIED RIGHT clause for variables greater than 32 767 characters.

IBM Extension

LIKE Clause

The LIKE clause allows you to define the PICTURE, USAGE, and SIGN characteristics of a data item by copying them from a previously defined data item. It also allows you to make the length of the data item you define different from the length of the original item.

Format



data-name
Can refer to an elementary item, a group item, or an index-name.

integer
Specifies the difference in length between the new and existing items.

It can be signed.

If a blank or a + precedes the integer, the new item is longer. If a – precedes the integer, the new item is shorter.

You cannot use the LIKE clause to:

- Change the length of an edited item
- Change the length of an index or pointer item
- Change the number of decimal places in a data item.

Note that an item whose attributes include BLANK WHEN ZERO is treated as an edited item.

The LIKE clause causes the new data item to inherit specific characteristics from the existing data item. These characteristics are the PICTURE, USAGE, SIGN, and BLANK WHEN ZERO attributes of the existing item.

The compiler generates comments to identify the characteristics of the new item. These comments appear after the statement containing the LIKE clause.

Note that the default USAGE IS DISPLAY and SIGN IS TRAILING characteristics do not print as comments.

The different USAGE clauses that you can specify for the original item result in a limited number of comments. Table 9 on page 133 illustrates this.

Table 9. Comments Generated based on Inherited USAGE Characteristics	
Inherited USAGE Clause	**Generated Comment**
PACKED-DECIMAL COMPUTATIONAL COMPUTATIONAL-3	* USAGE IS PACKED-DECIMAL
BINARY COMPUTATIONAL-4	* USAGE IS BINARY
INDEX	* USAGE IS INDEX
DISPLAY	This is the default usage, so a comment is not generated.
POINTER	* USAGE IS POINTER

The characteristics of the data item that you define using the LIKE clause are shown in the listing of your compiled program.

Rules and Restrictions

You can use the LIKE clause at level-numbers 01 through 49, and at level-number 77.

If you specify data-name or FILLER entries, you can put the LIKE clause in any position after them. Otherwise, you can put it in any position after the level-number.

You can specify one or more other clauses before or after the LIKE clause:

JUSTIFIED

SYNCHRONIZED

BLANK WHEN ZERO

VALUE

OCCURS.

Note that you can specify BLANK WHEN ZERO only if it has not previously been inherited.

You cannot use the LIKE clause with the following clauses:

REDEFINES

SIGN

USAGE

PICTURE.

If you specify any inherited clauses in the LIKE clause, a duplication error will result.

For numeric items, the total number of numeric characters in the new item cannot be zero. But if the item contains decimals, the number of characters in the integer portion can be zero.

If a PICTURE clause specifies a mixture of alphabetic, numeric, or alphanumeric characters, the new PICTURE clause specifies alphanumeric characters.

You cannot use the LIKE clause to define an item that is subordinate to the item that you name in the clause.

Examples

To create data item DEPTH with the same attributes as data item HEIGHT, you simply write:

```
DEPTH LIKE HEIGHT
```

To create data item PROVINCE with the same attributes as data item STATE, except one byte longer, you write:

```
PROVINCE LIKE STATE (+1)
```

For more detailed examples, see the *COBOL/400 User's Guide*.

End of IBM Extension

OCCURS Clause

The Data Division clauses that are used for table handling are the OCCURS clause and USAGE IS INDEX clause. For the USAGE IS INDEX description, see "USAGE Clause" on page 171.

IBM Extension

An item whose usage is POINTER can contain an OCCURS clause, or be subordinate to an item declared with an OCCURS clause.

Tables containing pointer data items are subject to pointer alignment as defined under "Pointer Alignment" on page 176. Where necessary, the compiler adds FILLER items to align the pointers in the first element of the table, plus a FILLER item at the end of the element to align the next pointer. This continues until all pointers in the table have been aligned.

End of IBM Extension

The OCCURS clause cannot appear in a data description entry whose level-number is 01, 66, 77, or 88.

Table Handling Concepts

Tables are often used in data processing. A table is a set of logically consecutive items, each of which has the same data description as the other items in the set. The items in a table can be described as separate contiguous items. However, this approach may not be satisfactory for two reasons. From a documentation standpoint, the homogeneity of the data items is not apparent; secondly, repetitive coding to reference unique data-names becomes a severe problem. Thus, a method of data reference is used which makes it possible to refer to all or to part of one table as an entity.

In COBOL, a table is defined with an OCCURS clause in its data description. The OCCURS clause specifies that the named item is to be repeated as many times as stated. The item so named is considered a table element, and its name and description apply to each repetition (or occurrence) of the item. Because the occurrences are not given unique data-names, reference to a particular occur-

rence can be made only by specifying the data-name of the table element. together with the occurrence number of the desired item within the element.

The occurrence number is known as a subscript and the technique of supplying the occurrence number of individual table elements is called subscripting. Subscripting is described in a subsequent section.

Table Handling Considerations

You should be aware of the following limitations when you work with tables:

- The number of occurrences of an item in the OCCURS clause can be up to a maximum of 3 000 000.
- Elementary table elements, including subordinate elements, have a size limit of 32 767 bytes. In the following example, if you specified `PIC X(512)` instead of `PIC X(511)`, the size of the element for level 1 would equal 32 768, and the table would not be created.

```
WORKING-STORAGE SECTION.
01  TABLE-NAME.
    05  LEVEL1     OCCURS 2 TIMES.
        10  LEVEL2      OCCURS 2 TIMES.
            15  LEVEL3        OCCURS 2 TIMES.
                20  LEVEL4         OCCURS 2 TIMES.
                    25  LEVEL5          OCCURS 2 TIMES.
                        30  LEVEL6         OCCURS 2 TIMES.
                            35  LEVEL7          OCCURS 2 TIMES
                                                PIC X(511).
```

  The compiler will ensure that the repetition factor in the PICTURE clause for table elements is no greater than 32 767 characters.

- You can use all COBOL statements (following the usual COBOL rules) with elements of large tables.
- You can use operands with a length of up to 3 000 000 bytes in the following statements:

  ADD CORRESPONDING
  CALL ... USING
  INITIALIZE
  MOVE (including MOVE CORRESPONDING)
  READ ... INTO
  RELEASE ... FROM
  RETURN ... INTO
  REWRITE ... FROM
  SEARCH (identifier-1 only)
  SUBTRACT CORRESPONDING
  WRITE ... FROM

Table Definition

The COBOL/400 compiler allows tables in one to seven dimensions.

To define a one-dimensional table, set up a group item that includes one OCCURS clause. Remember that the OCCURS clause cannot appear in a data description entry whose level-number is 01, 66, 77, or 88.

For example:

```
01 TABLE-ONE.
   05 ELEMENT-ONE OCCURS 3 TIMES.
      10 ELEMENT-A PIC X(4).
      10 ELEMENT-B PIC 9(4).
```

TABLE-ONE is the group item that contains the table. ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-A and ELEMENT-B are elementary items subordinate to ELEMENT-ONE.

To define a two-dimensional table, a one-dimensional table is defined within each occurrence of another one-dimensional table. For example:

```
01 TABLE-TWO.
   05 ELEMENT-ONE OCCURS 3 TIMES.
      10 ELEMENT-TWO OCCURS 3 TIMES.
         15 ELEMENT-A PIC X(4).
         15 ELEMENT-B PIC 9(4).
```

TABLE-TWO is the group item that contains the table. ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-TWO is an element of a two-dimensional table that occurs three times within each occurrence of ELEMENT-ONE. ELEMENT-A and ELEMENT-B are elementary items subordinate to ELEMENT-TWO.

To define a three-dimensional table, a one-dimensional table is defined within each occurrence of another one-dimensional table, which is itself contained within each occurrence of another one-dimensional table. For example:

```
01 TABLE-THREE.
   05 ELEMENT-ONE OCCURS 3 TIMES.
      10 ELEMENT-TWO OCCURS 3 TIMES.
         15 ELEMENT-THREE OCCURS 2 TIMES
            PICTURE X(8).
```

TABLE-THREE is the group item that contains the table. ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-TWO is an element of a two-dimensional table that occurs three times within each occurrence of ELEMENT-ONE. ELEMENT-THREE is an element of a three-dimensional table that occurs two times within each occurrence of ELEMENT-TWO. Figure 7 shows the storage layout for TABLE-THREE.

ELEMENT-ONE Occurs Three Times	ELEMENT-TWO Occurs Three Times	ELEMENT-THREE Occurs Two Times	Byte Displacement
			0
ELEMENT-ONE (1)	ELEMENT-TWO (1, 1)	ELEMENT-THREE (1, 1, 1)	
			8
		ELEMENT-THREE (1, 1, 2)	
			16
	ELEMENT-TWO (1, 2)	ELEMENT-THREE (1, 2, 1)	
			24
		ELEMENT-THREE (1, 2, 2)	
			32
	ELEMENT-TWO (1, 3)	ELEMENT-THREE (1, 3, 1)	
			40
		ELEMENT-THREE (1, 3, 2)	
			48
ELEMENT-ONE (2)	ELEMENT-TWO (2, 1)	ELEMENT-THREE (2, 1, 1)	
			56
		ELEMENT-THREE (2, 1, 2)	
			64
	ELEMENT-TWO (2, 2)	ELEMENT-THREE (2, 2, 1)	
			72
		ELEMENT-THREE (2, 2, 2)	
			80
	ELEMENT-TWO (2, 3)	ELEMENT-THREE (2, 3, 1)	
			88
		ELEMENT-THREE (2, 3, 2)	
			96
ELEMENT-ONE (3)	ELEMENT-TWO (3, 1)	ELEMENT-THREE (3, 1, 1)	
			104
		ELEMENT-THREE (3, 1, 2)	
			112
	ELEMENT-TWO (3, 2)	ELEMENT-THREE (3, 2, 1)	
			120
		ELEMENT-THREE (3, 2, 2)	
			128
	ELEMENT-TWO (3, 3)	ELEMENT-THREE (3, 3, 1)	
			136
		ELEMENT-THREE (3, 3, 2)	
			144

Figure 7. Storage Layout for TABLE-THREE

Table References

Whenever the user refers to a table element, or to any item associated with a table element, the reference must indicate which occurrence is intended.

For a one-dimensional table, the occurrence number of the desired element gives the complete information. For tables of more than one dimension, an occurrence number for each dimension must be supplied. In the three-dimensional table defined in the previous discussion, for example, a reference to ELEMENT-THREE must supply the occurrence number for ELEMENT-ONE, ELEMENT-TWO, and ELEMENT-THREE.

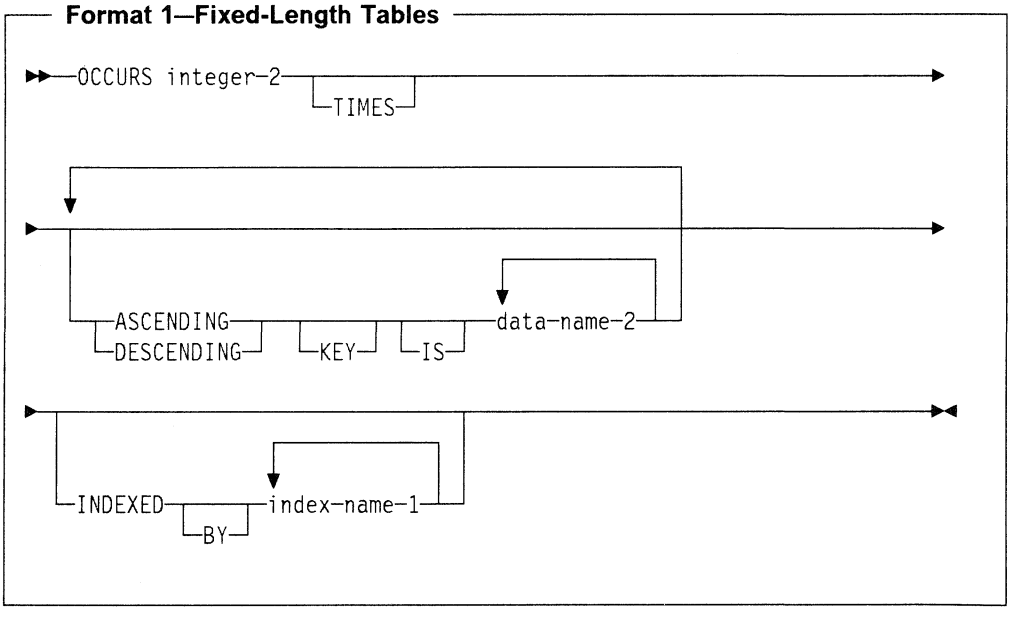
Formats for the OCCURS clause are:

- Fixed-length tables
- Variable-length tables.

Fixed-Length Tables

Fixed-length tables are specified using the OCCURS clause. Because seven subscripts or indexes are allowed, six nested levels and one outermost level of the Format 1 OCCURS clause are allowed. The Format 1 OCCURS clause may be specified as subordinate to the OCCURS DEPENDING ON clause. In this way, a table of up to seven dimensions may be specified.

Format 1—Fixed-Length Tables



integer-2

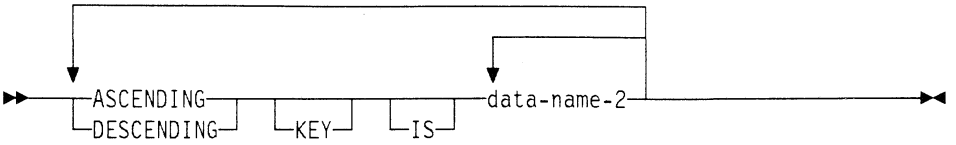
Specifies the exact number of occurrences, and must be greater than zero.

In the COBOL/400 language, **integer-2** must be between 1 and 3 000 000 bytes.

ASCENDING/DESCENDING KEY Phrase

Data is arranged in ascending or descending order (depending on the key word specified) according to the values contained in data-name-2. The data-names are listed in their descending order of significance.

Format



The order is determined by the rules for comparison of operands (see "Relation Condition" on page 192). The ASCENDING and DESCENDING KEY data items are used in OCCURS clauses and the SEARCH ALL statement for a binary search of the table element.

data-name-2

Must be the name of the subject entry, or the name of an entry subordinate to the subject entry.

If data-name-2 names the subject entry, that entire entry becomes the ASCENDING/DESCENDING KEY, and is the only key that may be specified for this table element.

If data-name-2 does not name the subject entry, then data-name-2:

- Must be subordinate to the subject of the table entry itself
- Must **not** be subordinate to, or follow, any other entry that contains an OCCURS clause
- Must not contain an OCCURS clause.

When the ASCENDING/DESCENDING KEY phrase is specified, the following rules apply:

- Keys must be listed in decreasing order of significance.
- You must arrange the data in the table in ASCENDING or DESCENDING sequence according to the collating sequence in use.
- A key may have DISPLAY, BINARY, PACKED-DECIMAL, or COMPUTATIONAL usage.

The following example illustrates the specification of ASCENDING KEY data items:

```
WORKING-STORAGE SECTION.
01  TABLE-RECORD.
  05  EMPLOYEE-TABLE OCCURS 100 TIMES
      ASCENDING KEY IS WAGE-RATE EMPLOYEE-NO
      INDEXED BY A,  B.
    10  EMPLOYEE-NAME                          PIC X(20).
    10  EMPLOYEE-NO                            PIC 9(6).
    10  WAGE-RATE                              PIC 9999V99.
    10  WEEK-RECORD OCCURS 52 TIMES
        ASCENDING KEY IS WEEK-NO INDEXED BY C.
      15  WEEK-NO                              PIC 99.
      15  AUTHORIZED-ABSENCES                  PIC  9.
      15  UNAUTHORIZED-ABSENCES                PIC  9.
      15  LATE-ARRIVALS                        PIC  9.
```

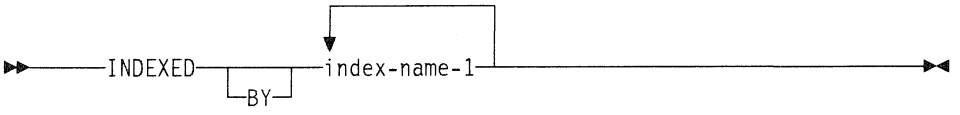
The keys for EMPLOYEE-TABLE are subordinate to that entry, while the key for WEEK-RECORD is subordinate to that subordinate entry.

In the preceding example, records in EMPLOYEE-TABLE must be arranged in ascending order of WAGE-RATE, and in ascending order of EMPLOYEE-NO within WAGE-RATE. Records in WEEK-RECORD must be arranged in ascending order of WEEK-NO. If they are not, results of any SEARCH ALL statement will be unpredictable.

INDEXED BY Phrase

The INDEXED BY phrase specifies the indexes that can be used with this table. The INDEXED BY phrase is required if indexing is used to refer to this table element. See "Subscripting Using Index-Names (Indexing)" on page 144.

Format



index-name-1

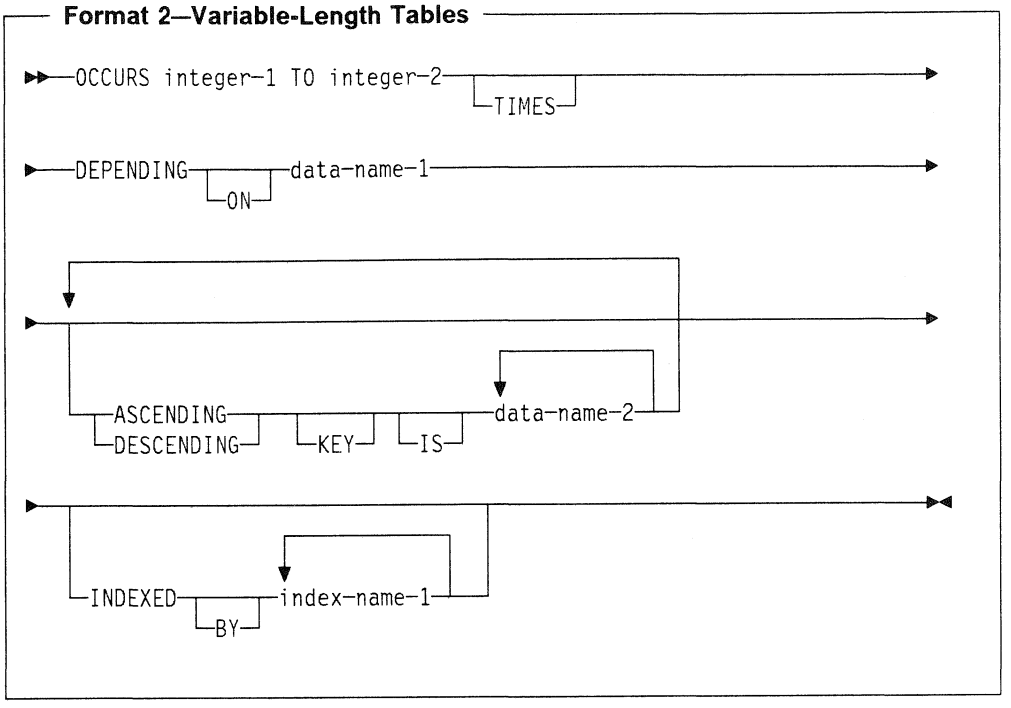
Must follow the rules for formation of user-defined words. At least one character must be alphabetic.

Each index-name specifies an index to be created by the compiler for use by the program. These index-names are **not** data-names, and are not identified elsewhere in the COBOL program; instead, they can be regarded as private special registers for the use of this object program only. As such, they are not data, or part of any data hierarchy; as such, each must be unique.

Variable-Length Tables

Variable-length tables are specified using the OCCURS DEPENDING ON clause.

Format 2—Variable-Length Tables



integer-1

The minimum number of occurrences.

The value of integer-1 must be greater than or equal to zero; it must also be less than the value of integer-2.

integer-2

The maximum number of occurrences.

The value of integer-2 must be no more than 32 767.

The **length** of the subject item is fixed; it is only the **number of repetitions** of the subject item that is variable.

OCCURS DEPENDING ON Clause

The OCCURS DEPENDING ON clause specifies variable-length tables.

data-name-1
Specifies the **object** of the OCCURS DEPENDING ON clause; that is, the data item whose current value represents the current number of occurrences of the subject item. The contents of items whose occurrence numbers exceed the value of the object are unpredictable.

The object of the OCCURS DEPENDING ON clause must not occupy any storage position within the range of any table (that is, any storage position from the first character position in the table through the last character position in the table).

At the time that the group item (or any data item that contains a subordinate OCCURS DEPENDING ON item or that follows but is not subordinate to the OCCURS DEPENDING ON item) is referenced, the value of the object referenced by data-name-1 must fall within the range integer-1 through integer-2.

When a group item containing a subordinate OCCURS DEPENDING ON item is referred to, the part of the table area used in the operation is determined as follows:

- If the object is outside the group, only that part of the table area that is specified by the object at the start of the operation will be used.
- If the object is included in the same group and the group data item is referenced as a sending item, only that part of the table area that is specified by the value of the object at the start of the operation will be used in the operation.
- If the object is included in the same group and the group data item is referenced as a receiving item, the maximum length of the group item will be used in the operation.

The **subject** of an OCCURS clause is the data-name of the data item containing the OCCURS clause. Except for the OCCURS clause itself, data description clauses used with the subject apply to each occurrence of the item described.

Subscripting or indexing is *required* whenever the subject is used in a statement other than SEARCH or USE FOR DEBUGGING, unless it is the object of a REDEFINES clause. In this case, the subject refers to one occurrence within a table element.

Subscripting and indexing are *not allowed* when the subject is used in a SEARCH or USE FOR DEBUGGING statement, or when it is the object of a REDEFINES clause. In this case, the subject represents an entire table element.

Note that the previous two restrictions do not apply to the LENGTH OF special register.

In one record description entry, any entry that contains an OCCURS DEPENDING ON clause may be followed only by items subordinate to it.

The OCCURS DEPENDING ON clause may not be specified as subordinate to another OCCURS clause.

All data-names used in the OCCURS clause may be qualified; they may not be subscripted or indexed.

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

- Has a level number of 01, 66, 77, or 88.
- Describes an item of variable size (an item is of variable size if any subordinate entry contains an OCCURS DEPENDING ON clause).
- Describes a redefined data item. (However, a redefined item can be subordinate to an item containing an OCCURS clause.) See "REDEFINES Clause" on page 158.

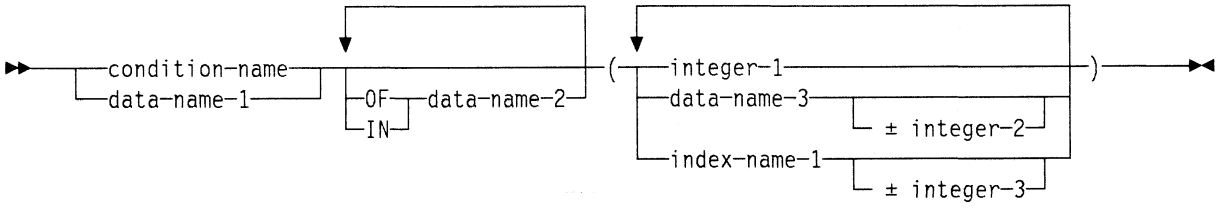
The ASCENDING/DESCENDING KEY and INDEXED BY clauses are described under "Fixed-Length Tables" on page 138.

Note: If you use the OCCURS DEPENDING ON clause, the table must contain no more than 32 767 occurrences, the length of a table element must be no more than 32 767 bytes, and the length of the whole table must be no more than 32 767 bytes.

Subscripting

Subscripting is a method of providing table references through the use of subscripts. A **subscript** is a positive integer whose value specifies the occurrence number of a table element.

Format



integer-1
May be signed. If signed, it must be positive.

data-name-3
Must be a numeric elementary item representing an integer. It may be qualified.

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

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. A data-name or index-name may be followed by either the operator + or the operator - and an integer (delimited by a space), which is used as an increment or decrement, respectively. It is permissible to mix integers, data-names, and index-names.

The subscripts, enclosed in parentheses, are written immediately following any qualification for the name of the table element. 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 considered a series of nested tables and the outermost table in the nest is considered to be the major table and the innermost table being the minor table, the subscripts are written from left to right in the order major, intermediate (up to five dimensions), and minor. For example, valid literal subscript references to TABLE-THREE are:

```
ELEMENT-THREE (1, 2, 1)

ELEMENT-THREE (2 2 1)
```

A reference to an item must not be subscripted unless the item is a table element or an item or condition-name associated with a table element.

The lowest permissible occurrence number represented by a subscript 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.

If the RANGE option is specified or implied, the system insures that the subscript value is valid. If the RANGE option is not active, it is **your** responsibility to ensure that the subscript value is valid. The RANGE option **does not** cause the system to verify that index entries are valid; it is **your** responsibility to ensure valid index values.

Note: See the *COBOL/400 User's Guide* for comprehensive information on the CRTCBLPGM command and the PROCESS statement.

Subscripting Using Integers or Data-Names

When an integer or data-name 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 or data-name may appear as the only subscript with one item and as one of two or more subscripts with another item. TABLE-THREE on page 136, assuming that SUB1, SUB2, and SUB3 are all items subordinate to SUBSCRIPT-ITEM are:

```
ELEMENT-THREE (SUB1 SUB2 SUB3)

ELEMENT-THREE IN TABLE-THREE (SUB1 OF SUBSCRIPT-ITEM, SUB2 OF
SUBSCRIPT-ITEM, SUB3 OF SUBSCRIPT-ITEM)
```

Subscripting Using Index-Names (Indexing)

Indexing facilitates such operations as table searching and manipulating specific items. To use indexing you assign 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 specifies the indexes that can be used with this table element. The INDEXED BY phrase is required if indexing is used to refer to this table element.

Each index-name must follow the rules for formation of a user-defined word; at least one character must be alphabetic. Each index-name specifies an index to

be created by the compiler for use by the program. These index-names are not data-names and are not identified elsewhere in the COBOL program; instead, they can be regarded as compiler generated registers for the use of this object program only. Therefore, they are not data or part of any data hierarchy; as such, each must be unique. An INDEX-NAME can only be referenced by a PERFORM, SET, or SEARCH statement, as a parameter in the USING phrase in a CALL statement, or in a relational condition comparison.

The initial value of an index at object time is undefined, and the index must be initialized before it is used as a subscript. 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 or data-name 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 by 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.

In **relative indexing**, there is an additional phrase 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 integer is considered to be an occurrence number, and is converted to an index value before being added to or subtracted from the index-name. 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 USAGE IS INDEX clause. The index value is moved to the index data item by the execution of a SET statement.

To be valid during execution, an index value must correspond to a table element occurrence of not less than one, nor greater than the highest permissible occurrence number.

Further information on index-names is given in the description of the INDEXED BY phrase of the OCCURS clause, on page 140.

Restrictions on Subscripting

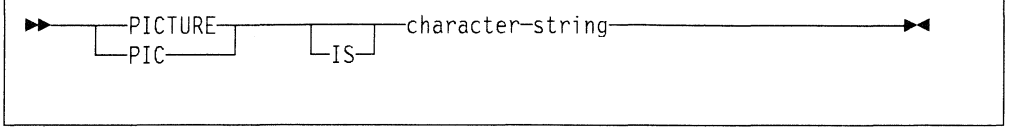
1. A data-name must not be subscripted or indexed when it is being used as a subscript or qualifier.
2. An index may be modified only by a PERFORM, SEARCH, or SET statement.
3. When a literal is used in a subscript, it must be a positive or unsigned integer.

4. When a literal is used in relative subscripting and indexing, it must be an unsigned integer.

PICTURE Clause

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

Format



The PICTURE clause must be specified for every elementary item except an index data item or the subject of the RENAMES clause. In these cases, use of this clause is prohibited.

The PICTURE clause may be specified only at the elementary level. PIC is an abbreviation for PICTURE and has the same meaning.

The PICTURE character-string is made up of certain COBOL characters used as symbols. The allowable combinations determine the category of the elementary data item.

The PICTURE character-string may contain a maximum of 30 characters.

DECIMAL-POINT IS COMMA, when specified in the SPECIAL-NAMES paragraph, exchanges the functions of the period and the comma in PICTURE character strings and in numeric literals.

The PICTURE clause is not allowed in descriptions of items described with USAGE IS INDEX or USAGE IS POINTER.

Symbols Used in the PICTURE Clause

The meaning of each PICTURE clause symbol is defined in Table 10 on page 147. The sequence in which PICTURE clause symbols must be specified is shown in Figure 8 on page 149. More detailed explanations of PICTURE clause symbols follow the figures.

Any punctuation character appearing within the PICTURE character-string is not considered a punctuation character, but rather a PICTURE character-string symbol.

The lowercase letters corresponding to the uppercase letters representing the PICTURE symbols A, B, P, S, V, X, Z, CR, and DB are equivalent to their uppercase representations in a PICTURE character-string.

Table 10 (Page 1 of 2). PICTURE Clause Symbol Meanings

Symbol	Meaning
A	A character position that can contain only a letter of the alphabet or a space.
B	A character position into which the space character is inserted.
P	An assumed decimal scaling position. It is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character P is not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric-edited items or in items that appear as arithmetic operands. The scaling position character P may appear only as a continuous string of Ps in the leftmost or rightmost digit positions within a PICTURE character-string. Because the scaling position character P implies an assumed decimal point (to the left of the Ps, if the Ps are leftmost PICTURE characters; to the right of the Ps, if the Ps are rightmost PICTURE characters), the assumed decimal point symbol, V, is redundant as either the leftmost or rightmost character within such a PICTURE description. 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 character 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: • Any operation requiring a numeric sending operand. • A MOVE statement where the sending operand is numeric and its PICTURE character-string contains the symbol P. • 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. • A comparison operation 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	An indicator of the presence (but not the representation nor, necessarily, the position) of an operational sign. It must be written as the leftmost character in the PICTURE string. An operational sign indicates whether the value of an item involved in an operation is positive or negative. The symbol S is not counted in determining the size of the elementary item, unless an associated SIGN clause specifies the SEPARATE CHARACTER phrase. Because hardware instructions use signs, you can improve performance by including the S in picture clauses whenever possible.
V	An indicator of the location of the assumed decimal point. It may appear only once in a character string. The V does not represent a character position and, therefore, is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string, the V is redundant.
X	A character position that may contain any allowable character from the EBCDIC character set.
Z	A leading numeric character position; when that position contains a zero, the zero is replaced by a space character. Each Z is counted in the size of the item.
9	A character position that contains a numeral and is counted in the size of the item.
1	IBM Extension A character position that contains a Boolean value of B"1" or B"0". Usage must be explicitly or implicitly defined as DISPLAY. End of IBM Extension
0	A character position into which the numeral zero is inserted. Each zero is counted in the size of the item.
/	A character position into which the slash character is inserted. Each slash is counted in the size of the item.

Table 10 (Page 2 of 2). PICTURE Clause Symbol Meanings

Symbol	Meaning
,	A character position into which a comma is inserted. This character is counted in the size of the item. If the comma insertion character 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.
.	An editing symbol that represents the decimal point for alignment purposes. In addition, it represents a character position into which a period is inserted. This character is counted in the size of the item. If the period 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. **Note:** For a given program, the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is specified 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	Editing sign control symbols. Each represents the character position into which the editing sign control symbol is placed. The symbols are mutually exclusive in one character string. Each character used in the symbol is counted in determining the size of the data item.
*	A check protect symbol—a leading numeric character position into which an asterisk is placed when that position contains a zero. Each asterisk (*) is counted in the size of the item.
$	A character position into which a currency symbol is placed. The currency symbol in a character string is represented either by the symbol $ or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division. The currency symbol is counted in the size of the item.

Figure 8 on page 149 shows the sequence in which PICTURE clause symbols must be specified. See the notes at the end of the figure.

Second Symbol \ First Symbol		Non-floating Insertion Symbols									Floating Insertion Symbols						Other Symbols						
		B	0	/	,	.	+ / −	+ / −	CR / DB	$	Z / *	Z / *	+ / −	+ / −	$	$	9	A / X	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	
	CR / DB	X	X	X	X	X				X	X	X			X	X	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	
	$	X	X	X	X		X								X								
	$	X	X	X	X	X	X								X	X				X		X	
Other Symbols	9	X	X	X	X	X	X			X	X		X		X		X	X	X	X		X	
	A / X	X	X	X													X	X					
	S																						
	V	X	X	X	X		X			X	X		X		X		X		X		X		
	P	X	X	X	X		X			X	X		X		X		X		X		X		
	P						X			X									X	X		X	
	1																						

Figure 8. PICTURE Clause Symbol Sequence

Notes to Figure 8 on page 149:

1. An X at an intersection indicates that the symbol(s) at the top of the column may, in a given character-string, appear anywhere to the left of the symbol(s) at the left of the row.
2. The $ character, however it is represented in the appropriate character set, is the default value for the currency symbol.
3. At least one of the symbols A, X, Z, 9, or *, or at least two of the symbols +, −, or $ must be present in a PICTURE string.
4. Nonfloating insertion symbols + and −, floating insertion symbols Z, *, +, −, and $, and the symbol P appear twice in the above PICTURE character precedence table. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of the symbol in the table represents its use to the right of the decimal point position.
5. Braces ({ }) indicate items that are mutually exclusive.

Character-String Representation

The following symbols may appear more than once in one PICTURE character-string:

```
A  B  P  X  Z  9  0  /  ,  +  −  *  $
```

An integer enclosed in parentheses immediately following any of these symbols specifies the number of consecutive occurrences of that symbol. The number of consecutive occurrences cannot exceed 3 000 000.

For example, the following two PICTURE clause specifications are equivalent:

```
PICTURE IS $99999.99CR

PICTURE IS $9(5).9(2)CR
```

The following symbols may appear only once in one PICTURE character-string:

```
S  V  .  CR  DB
```

or

IBM Extension

```
1 (A character position that contains a Boolean value).
```

End of IBM Extension

Each time any of the above symbols appears in the character-string, it represents an occurrence of that character or set of allowable characters in the data item.

Data Categories and PICTURE Rules

The allowable combinations of PICTURE symbols determine the data category of the item.

- Alphabetic items
- Numeric Items
- Numeric-edited items
- Alphanumeric items
- Alphanumeric-edited items

IBM Extension

- Boolean items.

End of IBM Extension

Alphabetic Items:

- The PICTURE character-string can contain only the symbol A.
- The contents of the item in standard data format must consist of any of the letters of the English alphabet and the space character.
- USAGE DISPLAY must be specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal containing only alphabetic characters or the figurative constant SPACE.

Numeric Items:

- Types of numeric items are:
  - Binary
  - Packed decimal (internal decimal)
  - Zoned decimal (external decimal).
- The PICTURE character-string can contain only the symbols 9, P, S, and V.
- The number of digit positions must range from 1 through 18, inclusive.
- If unsigned, the contents of the item in standard data format must contain a combination of the Arabic numerals 0-9. If signed, it may also contain a +, −, or other representation of the operational sign.
- The USAGE of the item can be DISPLAY, BINARY, COMPUTATIONAL, or PACKED-DECIMAL.

IBM Extension

- The USAGE of the item can be COMPUTATIONAL-3 or COMPUTATIONAL-4.

End of IBM Extension

- A VALUE clause associated with an elementary numeric item must specify a numeric literal or the figurative constant ZERO. A VALUE clause associated with a group item consisting of elementary numeric items must specify a **nonnumeric** literal or a figurative constant, because the group is considered

alphanumeric. In both cases, the literal is treated exactly as specified; no editing is performed.

Examples of numeric items:

PICTURE	Valid Range of Values
9999	0 through 9999
S99	-99 through +99
S999V9	-999.9 through +999.9
PPP999	0 through .000999
S999PPP	-1000 through -999000 and +1000 through +999000 or zero

Numeric-edited Items:

- The PICTURE character-string can contain the following symbols:

  B P V Z 9 0 / , . + - CR DB * $

  The combinations of symbols allowed are determined from the PICTURE clause symbol order allowed (see Figure 8 on page 149), and the editing rules (see "PICTURE Clause Editing" on page 153). The following additional rules also apply:

  - Either the BLANK WHEN ZERO clause must be specified for the item, or the string must contain at least one of the following symbols:

    B / Z 0 , . * + - CR DB $

  - The number of digit positions represented in the character-string must be in the range 1 through 18, inclusive.
  - The total length of the resultant character positions must be 127 or less.
- The contents of those character positions representing digits in standard data format must be one of the 10 Arabic numerals.
- USAGE DISPLAY must be specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal or a figurative constant. The literal is treated exactly as specified; no editing is done.

Alphanumeric Items:

- The PICTURE character-string must consist of either of the following:
  - The symbol X
  - Combinations of the symbols A, X, and 9.
    (A character-string containing all As or all 9s does not define an alphanumeric item.)
- The item is treated as if the character-string contained only the symbol X.
  - The contents of the item in standard data format may be any allowable characters from the EBCDIC character set.
  - USAGE DISPLAY must be specified or implied.
  - Any associated VALUE clause must specify a nonnumeric literal or a figurative constant.

Alphanumeric-edited Items:

- The PICTURE character-string can contain the following symbols:

  A X 9 B 0 /

- The string must contain at least one A or X, and at least one B or 0 (zero) or /.
- The contents of the item in standard data format may be any allowable character from the EBCDIC character set.
- The total length of the resultant character positions must be 127 or less.
- USAGE DISPLAY must be specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal or a figurative constant. The literal is treated exactly as specified; no editing is done.

IBM Extension

Boolean items: The following rules apply:

1. The PICTURE character-string can contain only the symbol 1.
2. Only one character 1 can be specified.
3. The USAGE of an item can only be DISPLAY.
4. An associated VALUE clause must specify a Boolean literal (B"1" or B"0") or zero.
5. The following clauses cannot be specified for a Boolean item:
   - SIGN clause
   - BLANK WHEN ZERO clause
   - ASCENDING/DESCENDING KEY clause.
6. The INDICATOR clause can be specified. (See the *COBOL/400 User's Guide* for more information about indicators.)

End of IBM Extension

PICTURE Clause Editing

There are two general methods of editing in a PICTURE clause:

- Insertion editing
  - Simple insertion
  - Special insertion
  - Fixed insertion
  - Floating insertion.
- Suppression and replacement editing
  - Zero suppression and replacement with asterisks
  - Zero suppression and replacement with spaces.

The type of editing allowed for an item depends on its **data category**. The type of editing that is valid for each category is shown below:

This type of editing is valid for alphabetic, alphanumeric edited, and numeric edited items. The valid insertion symbols for each category are shown in Table 12.

Table 11. Valid Editing for Each Data Category

Category	Type of Editing
Alphabetic	None
IBM Extension Boolean End of IBM Extension	None
Numeric	None
Alphanumeric	None
Alphanumeric edited	Simple insertion
Numeric edited	All

Simple Insertion Editing

This type of editing is valid for numeric-edited and alphanumeric-edited items.

Each insertion symbol is counted in the size of the item, and represents the position within the item where the equivalent characters will be inserted. Examples of simple insertion editing are shown in Table 13.

Table 12. Simple Insertion Editing — Valid Insertion Symbols for Each Data Category

Category	Valid Insertion Symbols
Alphabetic	None
IBM Extension Boolean End of IBM Extension	None
Numeric	None
Alphanumeric	None
Alphanumeric edited	B 0 /
Numeric edited	B 0 / ,

Table 13. Examples of Simple Insertion Editing

PICTURE	Value of Data	Edited Result
X(10)/XX	ALPHANUMER01	ALPHANUMER/01
X(5)BX(7)	ALPHANUMERIC	ALPHA NUMERIC
99,B999,B000	1234	01, 234, 000
99,999	12345	12,345

Special Insertion Editing

This type of editing is valid only for numeric-edited items.

The period (.) is the special insertion symbol; it also represents the actual decimal point for alignment purposes.

The period insertion symbol is counted in the size of the item, and represents the position within the item where the actual decimal point is inserted.

Either the actual decimal point or the symbol V as the assumed decimal point, but not both, must be specified in one PICTURE character-string.

Examples of special insertion editing:

PICTURE	Value of Data	Edited Results
999.99	1.234	001.23
999.99	12.34	012.34
999.99	123.45	123.45
999.99	1234.5	234.50

Fixed Insertion Editing

This type of editing is valid only for numeric-edited items. The following insertion symbols are used:

$ (currency symbol)

\+ – CR DB (editing-sign control symbols)

In fixed insertion editing, only one currency symbol and one editing sign control symbol can be specified in one PICTURE character-string.

Unless it is preceded by a + or – symbol, the currency symbol must be the first character in the character-string.

When either + or – is used as a symbol, it must be the first or last character in the character-string.

When CR or DB is used as a symbol, it must occupy the rightmost two character positions in the character-string. If these two character positions contain the symbols CR or DB, the uppercase letters are the insertion characters.

Editing sign control symbols produce results that depend on the value of the data item, as shown below:

Editing Symbol in PICTURE Character-String	Result: Data Item Positive or Zero	Result: Data Item Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

Examples of fixed insertion editing:

PICTURE	Value of Data	Edited Result
999.99+	+6555.556	555.55+
+9999.99	-6555.555	-6555.55
9999.99	+1234.56	1234.56
$999.99	-123.45	$123.45
-$999.99	-123.456	-$123.45
-$999.99	+123.456	$123.45
$9999.99CR	+123.45	$0123.45
$9999.99DB	-123.45	$0123.45DB

Floating Insertion Editing

This type of editing is valid only for numeric-edited items. The following symbols are used:

$ + –

Within one PICTURE character-string, these symbols are mutually exclusive as floating insertion characters.

Floating insertion editing is specified by using a string of at least two of the allowable floating insertion symbols to represent leftmost character positions into which these actual characters can be inserted.

The leftmost floating insertion symbol in the character-string represents the leftmost limit at which this actual character can appear in the data item. The rightmost floating insertion symbol represents the rightmost limit at which this actual character can appear.

The second leftmost floating insertion symbol in the character-string represents the leftmost limit at which numeric data can appear within the data item. Nonzero numeric data may replace all characters at or to the right of this limit.

Any simple-insertion symbols (B 0 / ,) within or to the immediate right of the string of floating insertion symbols are considered part of the floating character-string. If the period (.) special-insertion symbol is included within the floating string, it is considered to be part of the character-string.

In a PICTURE character-string, there are two ways to represent floating insertion editing and thus, two ways in which editing is performed:

1. Any or all leading numeric character positions to the left of the decimal point are represented by the floating insertion symbol. When editing is performed, a single floating insertion character is placed to the immediate left of the first nonzero digit in the data, or of the decimal point, whichever is farther to the left. The character positions to the left of the inserted character are filled with spaces.
2. All the numeric character positions are represented by the floating insertion symbol. When editing is performed, then:
   - If the value of the data is zero, the entire data item will contain spaces.
   - If the value of the data is nonzero, the result is the same as in rule 1.

To avoid truncation, the minimum size of the PICTURE character-string must be:

- The number of character positions in the sending item, plus
- The number of nonfloating insertion symbols in the receiving item, plus
- One character for the floating insertion symbol.

Examples of floating insertion editing:

PICTURE	Value of Data	Edited Result
$$$$.99	.123	$.12
$$$9.99	.12	$0.12
$,$$$,999.99	-1234.56	$1,234.56
+,+++,999.99	-123456.789	-123,456.78
$$,$$$,$$$.99CR	-1234567	$1,234,567.00CR
++,+++,+++.+++	0000.00	

Zero Suppression and Replacement Editing

This type of editing is valid only for numeric-edited items. In zero suppression editing, the symbols Z and * are used. These symbols are mutually exclusive in one PICTURE character-string.

The following symbols are mutually exclusive as floating replacement symbols in one PICTURE character-string:

Z * + – $

Specify zero suppression and replacement editing with a string of one or more of the allowable symbols to represent leftmost character positions in which zero suppression and replacement editing can be performed.

Any simple insertion symbols (B 0 / ,) within or to the immediate right of the string of floating editing symbols are considered part of the string. If the period (.) special insertion symbol is included within the floating editing string, it is considered to be part of the character-string.

In a PICTURE character-string, there are two ways to represent zero suppression, and two ways in which editing is performed:

- Any or all of the leading numeric character positions to the left of the decimal point are represented by suppression symbols. When editing is performed, any leading zero in the data that appears in the same character position as a suppression symbol is replaced by the replacement character. Suppression stops at the leftmost character:
  - That does not correspond to a suppression symbol
  - That contains nonzero data
  - That is the decimal point.
- All the numeric character positions in the PICTURE character-string are represented by the suppression symbols. When editing is performed, and the value of the data is nonzero, the result is the same as in the preceding rule. If the value of the data is zero, then:
  - If Z has been specified, the entire data item will contain spaces.
  - If * has been specified, the entire data item, except the actual decimal point, will contain asterisks.

Note: Do not specify both the asterisk (*) as a suppression symbol and the BLANK WHEN ZERO clause for the same entry.

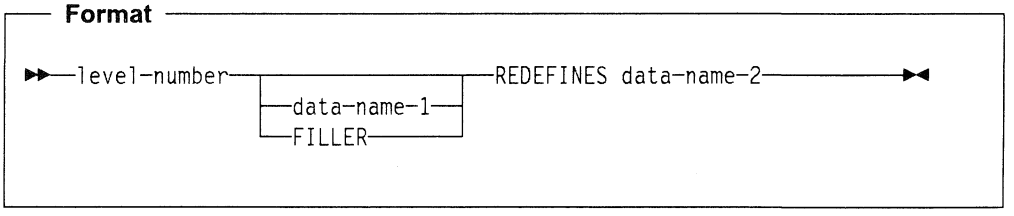
Examples of zero suppression and replacement editing:

PICTURE	Value of Data	Edited Result
****.**	0000.00	****.**
ZZZZ.ZZ	0000.00	
ZZZZ.99	0000.00	.00
****.99	0000.00	****.00
ZZ99.99	0000.00	00.00
Z,ZZZ.ZZ+	+123.456	123.45+
*,***.**+	-123.45	**123.45-
,*,***.**+	+12345678.9	12,345,678.90+
$Z,ZZZ,ZZZ.ZZCR	+12345.67	$ 12,345.67
$B*,***,***.**BBDB	-12345.67	$ ***12,345.67 DB

REDEFINES Clause

The REDEFINES clause allows you to use different data description entries to describe the same computer storage area.

Format



Note: Level-number, data-name-1, and FILLER are not part of the REDEFINES clause itself, and are included in the format only for clarity.

When specified, the REDEFINES clause must be the first entry following data-name-1 or FILLER. If data-name-1 or FILLER is not specified, the REDEFINES clause must be the first entry following the level-number.

The level-numbers of data-name-1 and data-name-2 must be identical, and must not be level 66 or level 88.

data-name-1
Identifies an alternate description for the same area, and is the **redefining** item or the **REDEFINES subject**.

data-name-2
Is the **redefined** item or the **REDEFINES object**.

When more that one level-01 entry is written subordinate to an FD entry, a condition known as implicit redefinition occurs. That is, the second level-01 entry implicitly redefines the storage allotted for the first entry. In such level-01 entries, the REDEFINES clause must not be specified.

Redefinition begins at data-name-1 and ends when a level-number less than or equal to that of data-name-1 is encountered. No entry having a level-number numerically lower than those of data-name-1 and data-name-2 may occur between these entries. For example:

```
05    A PICTURE X(6).
05    B REDEFINES A.
  10 B-1                    PICTURE X(2).
  10 B-2                    PICTURE 9(4).
05   C                      PICTURE 99V99.
```

In this example, A is the redefined item, and B is the redefining item. Redefinition begins with B and includes the two subordinate items B-1 and B-2. Redefinition ends when the level-05 item C is encountered.

The data description entry for the redefined item cannot contain an OCCURS clause. However, the redefined item may be subordinate to an item whose data description entry contains an OCCURS clause. In this case, the reference to the redefined item in the REDEFINES clause may not be subscripted. Neither the original definition nor the redefinition can include a variable occurrence data item.

Data-name-1, the redefining item, may be smaller than data-name-2, the redefined item. If the redefined item is specified with a level-number other than 01, however, the number of character positions it contains must be greater than or equal to the number of character positions in the redefining item.

One or more redefinitions of the same storage area are permitted. The entries giving the new descriptions of the storage area must be in the same section, and must immediately follow the description of the redefined area without intervening entries that define new character positions. Multiple redefinitions must all use the data-name of the original entry that defined this storage area. For example:

```
05    A                     PICTURE 9999.
05    B REDEFINES A         PICTURE 9V999.
05    C REDEFINES A         PICTURE 99V99.
```

The redefining entry (identified by data-name-1), and any subordinate entries, must not contain any VALUE clauses.

REDEFINES Clause Considerations

Data items within an area can be redefined without changing their lengths. For example:

```
05  NAME-2.
  10  SALARY                PICTURE XXX.
  10  SO-SEC-NO             PICTURE X(9).
  10  MONTH                 PICTURE XX.
05  NAME-1 REDEFINES NAME-2.
  10  WAGE                  PICTURE XXX.
  10  EMP-NO                PICTURE X(9).
  10  YEAR                  PICTURE XX.
```

Data item lengths and types can also be respecified within an area. For example:

```
05  NAME-2.
  10  SALARY                  PICTURE XXX.
  10  SO-SEC-NO               PICTURE X(9).
  10  MONTH                   PICTURE XX.
05  NAME-1 REDEFINES NAME-2.
  10  WAGE                    PICTURE 999V999.
  10  EMP-NO                  PICTURE X(6).
  10  YEAR                    PICTURE XX.
```

When an area is redefined, all descriptions of the area are always in effect; that is, redefinition does not cause any data to be erased and never supersedes a previous description. Thus, if B REDEFINES C has been specified, either of the two procedural statements, MOVE X TO B and MOVE Y TO C, could be executed at any point in the program.

In the first case, the area described as B would assume the value and format of X. In the second case, the same physical area (described now as C) would assume the value and format of Y. Note that, if the second statement is executed immediately after the first, the value of Y replaces the value of X in the one storage area.

The usage of a redefining data item need not be the same as that of a redefined item. This does not, however, cause any change in existing data. For example:

```
05  B                     PICTURE  99 USAGE DISPLAY VALUE 8.
05  C REDEFINES B         PICTURE S99 USAGE COMPUTATIONAL-4.
05  A                     PICTURE S99 USAGE COMPUTATIONAL-4.
```

The bit configuration of the DISPLAY value 8 is

```
1111 0000 1111 1000.
```

Redefining B does not change the bit configuration of the data in the storage area. Therefore, the following two statements produce different results:

```
ADD B TO A
ADD C TO A
```

In the first case, the value 8 is added to A (because B has USAGE DISPLAY). In the second statement, the value -48 is added to A (because C has USAGE COMPUTATIONAL-4) because the bit configuration (truncated to 2 decimal digits) in the storage area has the binary value -48.

The above example demonstrates how the improper use of redefinition may give unexpected or incorrect results.

REDEFINES Clause Examples

The REDEFINES clause may be specified for an item within the scope of an area being redefined (that is, an item subordinate to a redefined item). For example:

```
05  REGULAR-EMPLOYEE.
  10  LOCATION                        PICTURE A(8).
  10  GRADE                           PICTURE X(4).
  10  SEMI-MONTHLY-PAY                PICTURE 9999V99.
  10  WEEKLY-PAY REDEFINES SEMI-MONTHLY-PAY
                                      PICTURE 999V999.

05  TEMPORARY-EMPLOYEE REDEFINES REGULAR-EMPLOYEE.
  10  LOCATION                        PICTURE A(8).
  10  FILLER                          PICTURE X(6).
  10  HOURLY-PAY                      PICTURE 99V99.
```

The REDEFINES clause may also be specified for an item subordinate to a redefining item. For example:

```
05  REGULAR-EMPLOYEE.
  10  LOCATION                        PICTURE A(8).
  10  GRADE                           PICTURE X(4).
  10  SEMI-MONTHLY-PAY                PICTURE 999V999.

05  TEMPORARY-EMPLOYEE REDEFINES REGULAR-EMPLOYEE.
  10  LOCATION                        PICTURE A(8).
  10  FILLER                          PICTURE X(6).
  10  HOURLY-PAY                      PICTURE 99V99.
  10  CODE-H REDEFINES HOURLY-PAY     PICTURE 9999.
```

Undefined Results

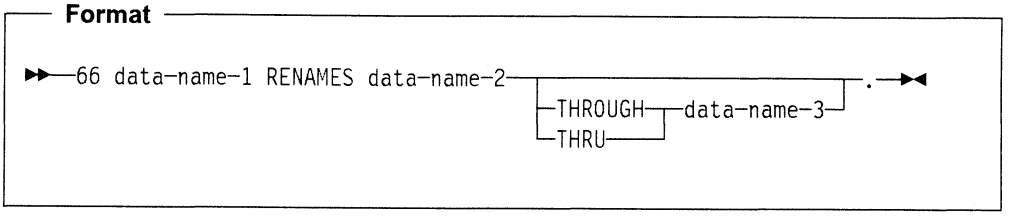
Undefined results may occur when:

- A redefining item is moved to a redefined item (that is, if `B REDEFINES C` and the statement `MOVE B TO C` is executed).
- A redefined item is moved to a redefining item (that is, if `B REDEFINES C` and if the statement `MOVE C TO B` is executed).

RENAMES Clause

The RENAMES clause specifies alternative, possibly overlapping, groupings of elementary data items.

Format



The special level-number 66 must be specified for data description entries that contain the RENAMES clause. Level-number 66 and data-name-1 are not part of the RENAMES clause itself, and are included in the format only for clarity.

One or more RENAMES entries can be written for a logical record. All RENAMES entries associated with one logical record must immediately follow that record's last data description entry.

data-name-1
Identifies an alternative grouping of data items.

A level-66 entry cannot rename a level-01, level-77, level-88, or another level-66 entry.

Data-name-1 cannot be used as a qualifier; it can be qualified only by the names of level indicator entries or level-01 entries.

data-name-2, data-name-3
Identify the original grouping of elementary data items; that is, they must name elementary or group items within the associated level-01 entry, and must not be the same data-name. Both data-names may be qualified.

The OCCURS clause must not be specified in the data entries for data-name-2 and data-name-3, or for any group entry to which they are sub-

ordinate. In addition, the OCCURS DEPENDING ON clause must not be specified for any item defined between data-name-2 and data-name-3.

When data-name-3 is specified, data-name-1 is treated as a group item that includes all elementary items:

- Starting with data-name-2 (if it is an elementary item) or the first elementary item within data-name-2 (if it is a group item)
- Ending with data-name-3 (if it is an elementary item) or the last elementary item within data-name-3 (if it is a group item)

The key words THROUGH and THRU are equivalent.

The leftmost character in data-name-3 must not precede that in data-name-2; the rightmost character in data-name-3 must follow that in data-name-2. This means that data-name-3 cannot be subordinate to data-name-2.

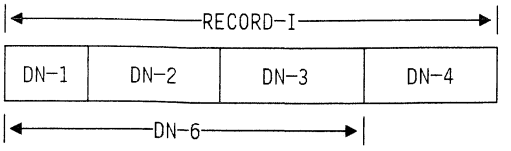
When data-name-3 is not specified, all of the data attributes of data-name-2 become the data attributes for data-name-1. That is:

- When data-name-2 is a group item, data-name-1 is treated as a group item.
- When data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

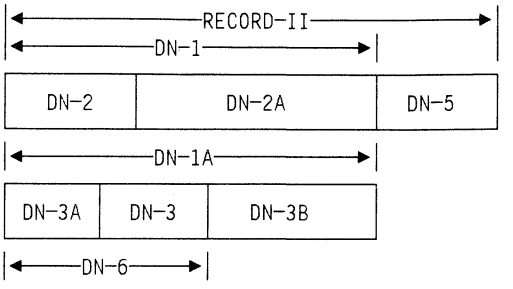
Figure 9 illustrates valid and invalid RENAMES clause specifications.

COBOL Specifications | **Storage Layouts**

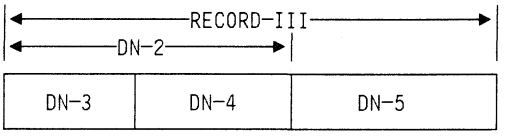
```
Example 1 (Valid)
  01   RECORD-I.
       05 DN-1... .
       05 DN-2... .
       05 DN-3... .
       05 DN-4... .
  66   DN-6 RENAMES DN-1 THROUGH DN-3.
```




```
Example 2 (Valid)
  01   RECORD-II.
       05 DN-1.
          10 DN-2... .
          10 DN-2A... .
       05 DN-1A REDEFINES DN-1.
          10 DN-3A... .
          10 DN-3... .
          10 DN-3B... .
       05 DN-5... .
  66   DN-6 RENAMES DN-2 THROUGH DN-3.
```

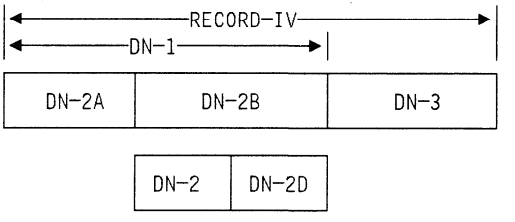



```
Example 3 (Invalid)
  01   RECORD-III.
       05 DN-2.
          10 DN-3... .
          10 DN-4... .
       05 DN-5... .
  66   DN-6 RENAMES DN-2 THROUGH DN-3.
```



DN-6 is indeterminate

```
Example 4 (Invalid)
  01   RECORD-IV.
       05 DN-1.
          10 DN-2A... .
          10 DN-2B... .
          10 DN-2C REDEFINES DN-2B.
             15 DN-2... .
             15 DN-2D... .
       05 DN-3... .
  66   DN-4 RENAMES DN-1 THROUGH DN-2.
```



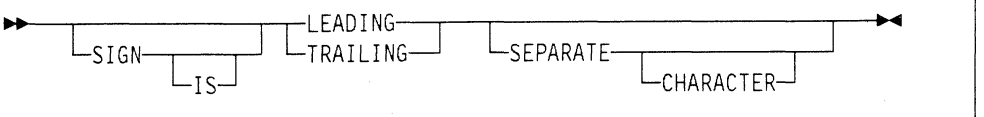
DN-4 is indeterminate

Figure 9. RENAMES Clause—Valid and Invalid Specifications

SIGN Clause

The SIGN clause specifies the position and mode of representation of the operational sign for a numeric entry.

Format



The SIGN clause may be specified only for a signed numeric data description entry (that is, one whose PICTURE character-string contains an S), or for a group item that contains at least one such elementary entry. USAGE IS DISPLAY must be specified, explicitly or implicitly.

If a SIGN clause is specified in either an elementary or group entry subordinate to a group item for which a SIGN clause is specified, the SIGN clause for the subordinate entry takes precedence for the subordinate entry.

If you specify the CODE-SET clause in an FD entry, any signed numeric data description entries associated with that file description entry must be described with the SIGN IS SEPARATE clause.

The SIGN clause is required only when an explicit description of the properties and/or position of the operational sign is necessary.

When specified, the SIGN clause defines the position and mode of representation of the operational sign for the numeric data description entry to which it applies, or for each signed numeric data description entry subordinate to the group to which it applies.

If the SEPARATE CHARACTER phrase is not specified, then:

- The operational sign is presumed to be associated with the LEADING or TRAILING digit position, whichever is specified, of the elementary numeric data item. (In this instance, specification of SIGN IS TRAILING is the equivalent of the standard action of the compiler.)
- The character S in the PICTURE character string is not counted in determining the size of the item (in terms of standard data format characters).

If the SEPARATE CHARACTER phrase is specified, then:

- The operational sign is presumed to be the LEADING or TRAILING character position, whichever is specified, of the elementary numeric data item. This character position is not a digit position.
- The character S in the PICTURE character string is counted in determining the size of the data item (in terms of standard data format characters).
- + is the character used for the positive operational sign.
- - is the character used for the negative operational sign.

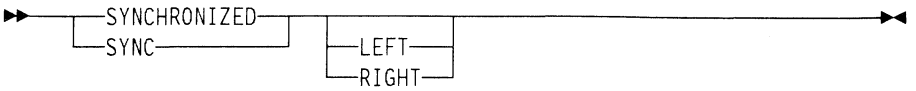
Every numeric data description entry whose PICTURE contains the symbol S is a signed numeric data description entry. If the SIGN clause is also specified for

such an entry, and conversion is necessary for computations or comparisons, the conversion takes place automatically.

SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the alignment of an elementary item on a natural boundary in storage.

Format



SYNC is an abbreviation for SYNCHRONIZED and has the same meaning.

The SYNCHRONIZED clause is never required, but may improve performance on some systems for binary items used in arithmetic.

The SYNCHRONIZED clause may appear only at the elementary level. It is not permitted for group items.

IBM Extension

The SYNCHRONIZED clause is implicit for pointer data items.

End of IBM Extension

Depending on the USAGE that is specified for an item, the SYNCHRONIZED clause has a particular effect. Figure 10 on page 167 shows how the USAGE of an item determines the effect of the SYNCHRONIZED clause upon it.

If the USAGE is...	The SYNCHRONIZED clause...
DISPLAY	is syntax checked but does not affect execution
PACKED-DECIMAL	is syntax checked but does not affect execution
COMPUTATIONAL-3	is syntax checked but does not affect execution
BINARY: PIC S9(1) thru S9(4)	aligns data item at a multiple of 2 relative to the beginning of the record
PIC S9(5) thru S9(18)	aligns data item at a multiple of 4 relative to the beginning of the record
COMPUTATIONAL-4	functions the same as for USAGE BINARY
COMPUTATIONAL	is syntax checked but does not affect execution
INDEX	is not permitted
POINTER	aligns data item at a multiple of 16 relative to the beginning of the record

Figure 10. Data item USAGE and the SYNCHRONIZED clause

LEFT
Specifies that the elementary item is to be positioned so that it will begin at the left character position of the natural boundary in which the elementary item is placed.

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

When specified, the LEFT and the RIGHT phrases are syntax-checked, but they have no effect on the execution of the program.

The length of an elementary item is not affected by the SYNCHRONIZED clause.

When the SYNCHRONIZED clause is specified for an item that also contains a REDEFINES clause, the data item that is redefined must have the proper boundary alignment for the data item that redefines it. For example, if you write the following, be sure that data item A begins at a multiple of 4 bytes relative to the beginning of the record:

```
02 A                    PICTURE X(4).
02 B REDEFINES A        PICTURE S9(9) BINARY SYNC.
```

When the SYNCHRONIZED clause is specified for a binary item that is the first elementary item subordinate to an item that contains a REDEFINES clause, the item must not require the addition of unused character positions.

If, as a result of alignment, a data item is not adjacent to its immediately preceding elementary item, an implicit FILLER data item is generated. This FILLER item is treated as if it were an item with a level number equal to that of the preceding item. The size of this implicit FILLER item is calculated as follows:

- The total number of characters occupied by all elementary data items preceding the aligned item are added together, including any implicit FILLER items previously added.
- This sum is divided by the factor `m` used as a multiplier in the above calculation of alignment (2, 4, or 16).
- If the remainder `r` of this division is equal to zero, no implicit FILLER item is required. If the remainder is not equal to zero, the size of the implicit FILLER item is equal to `m - r`.

The size of the implicit FILLER item is included in the size of any group item that contains it.

Group items are naturally defined as alphanumeric. Any FILLER items are initialized with spaces. Implicit FILLER items generated through the SYNCHRONIZED clause, then, are also initialized with spaces under the (default) *STDINZ compiler option. Under the *NOSTDINZ option, these implicit FILLER items will contain hexadecimal zeroes.

An implicit FILLER item may also be added by the compiler when a group item is defined with an OCCURS clause and contains data items that are subject to alignment. To determine whether an implicit FILLER is to be added, the following action is taken:

- The compiler calculates the size of the group item, including all necessary implicit FILLER items.
- This sum is divided by the largest `m` required by any elementary item within the group.
- If `r` is equal to zero, no implicit FILLER item is required. If `r` is not equal to zero, an implicit FILLER item of size `m - r` must be added.

An implicit FILLER item may be inserted at the end of each occurrence of the group item containing the OCCURS clause. This is done to synchronize subsequent occurrences.

Items at the group level will be boundary aligned according to the following rules:

Area	Level Number	Boundary Alignment
Working-Storage Section	01 77	16 bytes 8 bytes IBM Extension 77-level items that are involved in pointer moves are aligned on 16-byte boundaries. End of IBM Extension
File Section	01	Compiler assumes a 16-byte boundary for synchronizing items.
Linkage Section	01 77	Compiler assumes a 16-byte boundary for synchronizing items. It is the user's responsibility to ensure that the input parameters passed are also synchronized.

Synchronization: If a data item description contains the SYNCHRONIZED clause, then the position of that item is aligned such that the right-hand (least-significant) end is on the natural boundary. Extra storage is reserved adjacent to synchronized items to achieve this alignment; these bytes, known as padding bytes or implicit FILLER bytes, are accessible to the COBOL program.

Each elementary data item that is described as SYNCHRONIZED is aligned to the natural storage boundary that corresponds to its data item storage assignment. Thus, a numeric data item with a PICTURE description of S9(5) would be assigned 4 bytes of storage (1 padding byte and 3 data bytes) and, if SYNCHRONIZED was specified, would be aligned to the next nearest 4-byte boundary (with the total (4-byte) storage assignment aligned such that the number of bytes from the beginning of the record containing that item to the left-hand (most-significant) end of that item was a multiple of four). If the previous item does not end on a 4-byte boundary, then implicit FILLER assignments are necessary to achieve this.

Other such implicit FILLER bytes may be generated by the use of SYNCHRONIZED items in non-elementary data descriptions containing an OCCURS clause. This is because further extra bytes may need to be reserved for each group item occurrence in order that the second or following occurrences have the same alignment to the natural boundaries of the computer storage as did the first occurrence.

Example of Implicit FILLER: The following COBOL data description will produce the computer storage allocation shown in Figure 11.

```
01 UNSYNCHRONIZED-RECORD
     02 UNSYNCHRONIZED-DATA-1 PIC 9(3) DISPLAY.
     02 UNSYNCHRONIZED-DATA-2 PIC X(2).
01 COMPOUND-REPEATED-RECORD.
     02 ELEMENTARY-ITEM-1 PIC X (2).
     02 GROUP-ITEM OCCURS 3 TIMES.
        03 ELEMENTARY-ITEM-2 PIC X.
        03 ELEMENTARY-ITEM-3 PIC S9(2) BINARY SYNC.
        03 ELEMENTARY-ITEM-4 PIC S9(4) V9(2) BINARY SYNC.
        03 ELEMENTARY-ITEM-5 PIC X (5).
```

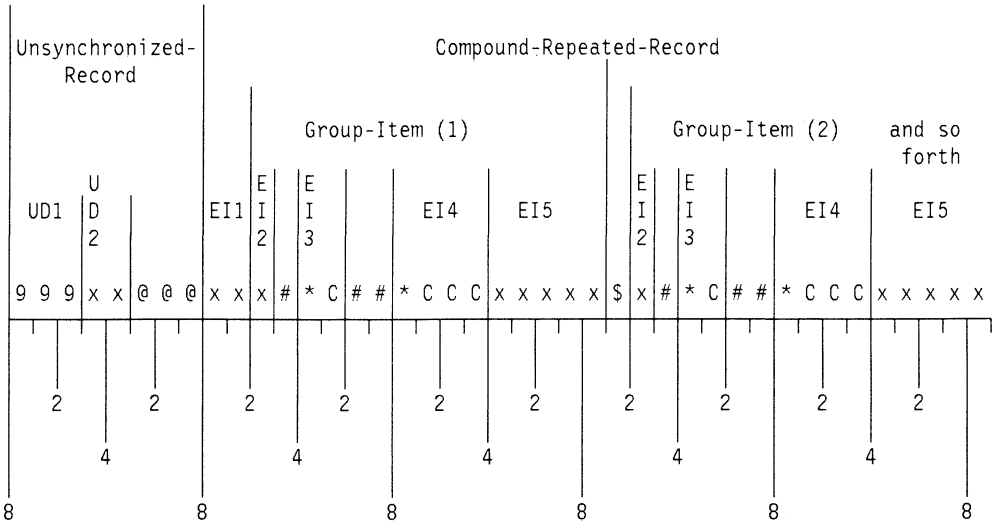


Figure 11. Computer Storage Allocation

@ Indicates implicit FILLER bytes allocated because of automatic synchronization or a record (01-level) description

# Indicates implicit FILLER bytes allocated when the following data item is explicitly synchronized

***** The first byte of a BINARY item that has been synchronized

$ Indicates implicit FILLER bytes allocated when a non-elementary item is subject to an OCCURS clause.

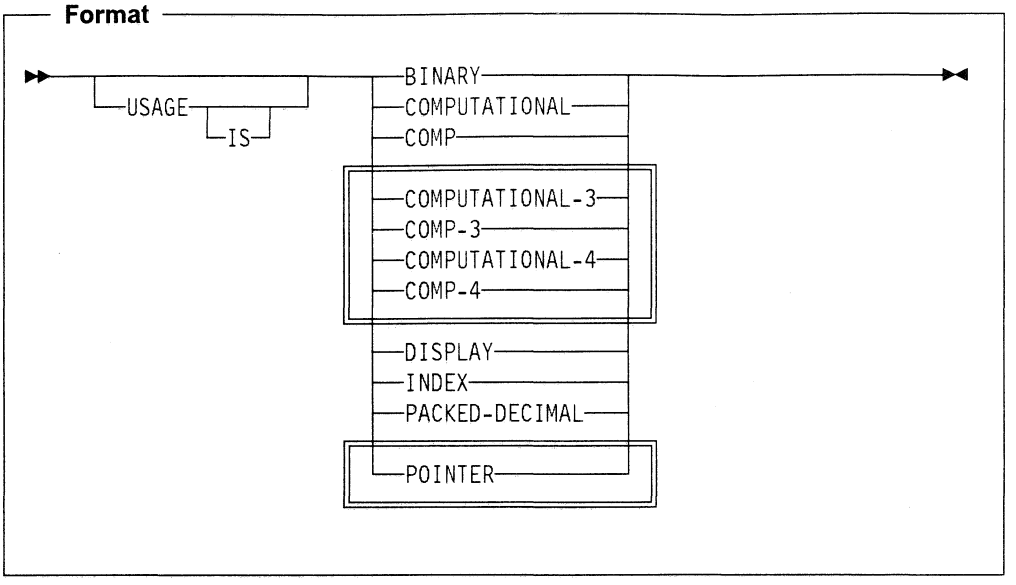
9 Indicates bytes allocated for a numeric DISPLAY character

X Indicates bytes allocated for an alphanumeric DISPLAY character

C Indicates bytes allocated for a BINARY data storage

USAGE Clause

Format



The USAGE clause can be specified for an entry at any level. However, if it is specified at the group level, it applies to each elementary item in the group. The usage of an elementary item must not contradict the usage of a group to which the elementary item belongs.

The USAGE clause specifies the format in which data is represented in storage. The format may be restricted if certain Procedure Division statements are used.

When the USAGE clause is not specified at either the group or elementary level, it is assumed that the usage is DISPLAY.

Computational Items

A computational item is a value used in arithmetic operations. It must be numeric. If the USAGE of a group item is described with any of these items, the elementary items within the group have this usage. The group itself is considered nonnumeric and cannot be used in numeric operations, except for those using the CORRESPONDING phrase (see "CORRESPONDING Phrase" on page 209).

The maximum length of a computational item is 18 decimal digits.

The PICTURE of a computational item may contain only:

9 One or more numeric character positions
S One operational sign
V One implied decimal point
P One or more decimal scaling positions.

BINARY

Specified for binary data items. Such items have a decimal equivalent consisting of the decimal digits 0 through 9, plus a sign. Negative numbers are represented as the two's complement of the positive number with the same absolute value.

The amount of storage occupied by a binary item depends on the number of decimal digits defined in its PICTURE clause:

Digits in PICTURE Clause	Storage Occupied
1 through 4	2 bytes
5 through 9	4 bytes
10 through 18	8 bytes

The leftmost bit of the storage area is the operational sign.

For better performance, avoid using 8-byte binary items. You can specify these items for convenience, but the compiler must make conversions in order to use them.

PACKED-DECIMAL
Specified for internal decimal items. Such an item appears in storage in packed decimal format. There are 2 digits for each character position, except for the trailing character position, which is occupied by the low-order digit and the sign. Such an item may contain any of the digits 0 through 9, plus a sign, representing a value not exceeding 18 decimal digits.

The sign representation is shown in Figure 12 on page 174.

COMPUTATIONAL or COMP
Specified for internal decimal items. Such an item appears in storage as 2 digits per byte, with the sign contained in the 4 rightmost bits of the rightmost byte. An internal decimal item can contain any of the digits 0 through 9 plus a sign. If the PICTURE of an internal decimal item does not contain an S, the sign position is occupied by a bit configuration that is interpreted as positive.

IBM Extension

COMPUTATIONAL-3 or COMP-3 (internal decimal)
This is the equivalent of PACKED-DECIMAL.

COMPUTATIONAL-4 or COMP-4 (binary)
This is the equivalent of BINARY.

End of IBM Extension

To improve compilation performance, specify odd numbers of numeric character positions in the picture clauses for COMP-3 (packed decimal) items. Internally, the rightmost byte of a packed decimal item contains a digit and a sign, and any other bytes contain two digits. If you use the more efficient configuration, the compiler does not need to supply the missing digit.

DISPLAY Phrase

The data item is stored in character form, 1 character for each 8-bit byte. This corresponds to the format used for printed output. DISPLAY can be explicit or implicit.

USAGE IS DISPLAY is valid for the following types of items:

- Alphabetic
- Alphanumeric
- Alphanumeric-edited
- Numeric-edited

IBM Extension

- Boolean

End of IBM Extension

- External decimal (numeric).

Alphabetic, **alphanumeric**, **alphanumeric-edited**, and **numeric-edited** items are discussed in "Data Categories and PICTURE Rules" on page 151.

The PICTURE character-string of a zoned item can contain only 9s, the operational sign symbol S, the assumed decimal point V, and one or more Ps.

External Decimal Items are sometimes referred to as **zoned decimal** items. Each digit of a number is represented by a single byte. The 4 high-order bits of each byte are zone bits; the 4 high-order bits of the low-order byte represent the sign of the item. If the number is positive, these four bits contain a hexadecimal F. If the number is negative, these four bits contain a hexadecimal D. The 4 low-order bits of each byte contain the value of the digit.

The maximum length of an external decimal item is 18 digits.

The PICTURE character-string of an external decimal item may contain only 9s; the operational-sign, S; the assumed decimal point, V; and one or more Ps.

Examples of external decimal items are shown in Figure 12 on page 174.

ITEM	DESCRIPTION	VALUE	INTERNAL REPRESENTATION*
External Decimal	PIC S9999 DISPLAY	+1234	F1 F2 F3 F4
		−1234	F1 F2 F3 D4
		1234	F1 F2 F3 F4
	PIC 9999 DISPLAY	+1234	F1 F2 F3 F4
		−1234	F1 F2 F3 F4
		1234	F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN LEADING	+1234	F1 F2 F3 F4
		−1234	D1 F2 F3 F4
		1234	F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN TRAILING SEPARATE	+1234	F1 F2 F3 F4 4E
		−1234	F1 F2 F3 F4 60
		1234	F1 F2 F3 F4 4E
	PIC S9999 DISPLAY SIGN LEADING SEPARATE	+1234	4E F1 F2 F3 F4
		−1234	60 F1 F2 F3 F4
		1234	4E F1 F2 F3 F4
Internal Decimal	PIC S9999 {COMP }	+1234	01 23 4F
	{COMP-3}	−1234	01 23 4D
	PIC 9999 {COMP }	+1234	01 23 4F
	{COMP-3}	−1234	01 23 4F
Binary	PIC S9999 COMP-4	+1234	04 D2
		−1234	FB 2E
	PIC 9999 COMP-4	+1234	04 D2
		−1234	04 D2

* The internal representation of each byte is shown as two hex digits.
The bit configuration for each digit is as follows:

Hex Digit	Bit Configuration	Hex Digit	Bit Configuration
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

NOTES:
1. The leftmost bit of a binary number represents the sign: 0 is positive, 1 is negative.
2. Negative binary numbers are represented in twos complement form.
3. Hex 4E represents the EBCDIC character +, Hex 60 represents the EBCDIC character −.
4. Specification of SIGN TRAILING (without the SEPARATE CHARACTER option) is the equivalent of the standard action of the compiler.

Figure 12. Internal Representation of Numeric Items

INDEX Phrase

A data item defined with the INDEX phrase is an **index data item**.

An **index data item** is a 4-byte elementary item (not necessarily connected with any table) that can be used to save index-name values for future reference. Through a SET statement, an index data item can be assigned an index-name value.

The index-name value is the displacement, which corresponds to an occurrence number in the table. The index-name value equals:

```
(occurrence-number — 1) * entry length
```

Direct references to an index data item can be made only in a SEARCH statement, a SET statement, a relation condition, the USING phrase of the Procedure Division header, or the USING phrase of the CALL statement.

An index data item can be part of a group item referred to in a MOVE statement or an input/output statement.

An index data item saves values that represent table occurrences, yet is not necessarily defined as part of any table. Thus, when it is referred to directly in a SEARCH or SET statement, or indirectly in a MOVE or input/output statement, there is no conversion of values when the statement is executed.

The USAGE IS INDEX clause may be written at any level. If a group item is described with the USAGE IS INDEX clause, the elementary items within the group are index data items; the group itself is not an index data item, and the group name may not be used in SEARCH and SET statements or in relation conditions. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

An index data item cannot be a conditional variable.

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

IBM Extension

POINTER Phrase

A data item defined with the USAGE IS POINTER clause is a **pointer data item**.

A pointer data item is a 16-byte elementary item that can be used to accomplish limited base addressing. Pointer data items can be compared for equality, or moved to other pointer items.

A pointer data item may only be used in:

- A SET statement (Format 5 only)
- A relation condition
- The USING phrase of a CALL statement or Procedure Division header
- Expressions involving ADDRESS OF or LENGTH OF.

The USAGE IS POINTER clause may be written at any level except 66 or 88.

If a group item is described with the USAGE IS POINTER clause, the elementary items within the group are pointer data items. The group itself, however, is not a pointer data item and cannot be used in the syntax where a pointer data item is allowed.

Pointer data items can be part of a group that is referred to in a MOVE statement or an I/O statement. If, however, a pointer data item is part of a group, there is

no conversion of pointer values to another internal representation when the statement runs.

A pointer data item can be the subject or object of a REDEFINES clause.

A VALUE clause for a pointer data item can contain NULL or NULLS only.

A pointer data item does not belong to a class or category, and it cannot be used as a conditional variable.

The JUSTIFIED, PICTURE, SIGN, and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items defined with the USAGE IS POINTER clause.

Pointer data items are ignored in CORRESPONDING operations.

A pointer data item can be written to a file, but if you later read the record containing the pointer data item, the item will no longer represent a valid address.

Note: USAGE IS POINTER is implicitly specified for the ADDRESS OF special register.

Note that you cannot treat COBOL/400 pointer data items as ordinary numbers.

Pointer Alignment

When a pointer data item is referenced, or is the subject of a REDEFINES clause, the object item must be in **alignment**. In other words, it must be located at an offset that is a multiple of 16 bytes from the beginning of the record.

A data item described with USAGE IS POINTER in the Working-storage or File section will be aligned. If the pointer data item is part of a structure that begins at level-number 01, the compiler aligns the beginning of the structure. After that, the compiler puts FILLER items in front of the pointer data item to make sure that it is also in alignment. The compiler issues a warning when it adds these FILLER items.

In the Linkage section, the compiler does not add FILLER items to the structure, but it issues warnings regarding its assumption that you have aligned the 01-level items.

If a pointer is the subject of a REDEFINES clause in the Linkage section, and the object of the clause is not a pointer, you will receive a warning that you need to maintain pointer alignment. For the same situation in the Working-storage or File section, an error will result if you do not align the object of the clause.

You can specify the SYNCHRONIZED clause along with USAGE IS POINTER, but this clause is already implicit for pointer data items.

If the pointer data item is part of a table, the first item in the table is aligned, and to make sure that all occurrences of the pointer data item are also aligned, a filler item might be added to the end of the table. When 77 items are involved in pointer moves they are forced into alignment.

To avoid adding FILLER items to data structures, place pointer data items at the beginnings of the structures.

End of IBM Extension

VALUE Clause

The VALUE clause specifies the initial contents of a data item or the value(s) associated with a condition-name.

The use of the VALUE clause differs depending on the Data Division section in which it is specified.

In the File and Linkage sections, the VALUE clause must be used only in condition-name entries. In the Working-Storage Section, the VALUE clause may be used in condition-name entries, or in specifying the initial value of any data item. The data item assumes the specified value at the beginning of program execution. If the initial value is not explicitly specified, it is unpredictable.

Rules for Literal Values:

- Wherever a literal is specified, a figurative constant may be substituted.
- If the item is numeric, all VALUE clause literals must be numeric. If the literal defines the value of a Working-Storage item, the literal is aligned according to the rules for numeric moves, with one additional restriction: The literal must not have a value that requires truncation of nonzero digits. If the literal is signed, the associated PICTURE character-string must contain a sign symbol (S).
- All numeric literals in a VALUE clause of an item must have a value that is within the range of values indicated by the PICTURE clause for that item. For example, for a PICTURE of 99PPP, the literal must fall within the range of 1 000 through 99 000, or it must be zero. For a PICTURE of PPP99, the literal must fall within the range of 0.000 00 through 0.000 99.
- If the item is a group item, or an elementary alphabetic, alphanumeric, alphanumeric-edited, or numeric-edited item, all VALUE clause literals must be nonnumeric literals. The literal is aligned according to the alignment rules, with one additional restriction: the number of characters in the literal must not exceed the size of the item.

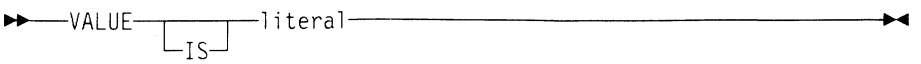
IBM Extension

If the item is Boolean, the VALUE clause must be a Boolean literal.

End of IBM Extension

- The functions of the editing characters in a PICTURE clause are ignored in determining the initial appearance of the item described. However, editing characters are included in determining the size of the item. Therefore, any editing characters must be included in the literal. For example, if the item is defined as PICTURE +999.99 and the value is to be +12.34, then the VALUE clause should be specified as VALUE '+012.34'.
- A maximum of 32 767 bytes can be initialized by means of a single VALUE clause. A maximum of 65 472 bytes can be initialized by **all** of the VALUE clauses contained within a single program.

Format 1—Literal Value



Format 1 specifies the initial value of a data item. Initialization is independent of any BLANK WHEN ZERO or JUSTIFIED clause specified.

A format 1 VALUE clause specified in a data description entry that contains or is subordinate to an OCCURS clause causes every occurrence of the associated data item to be assigned the specified value.

The VALUE clause must not be specified for a data description entry that contains, or is subordinate to, an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

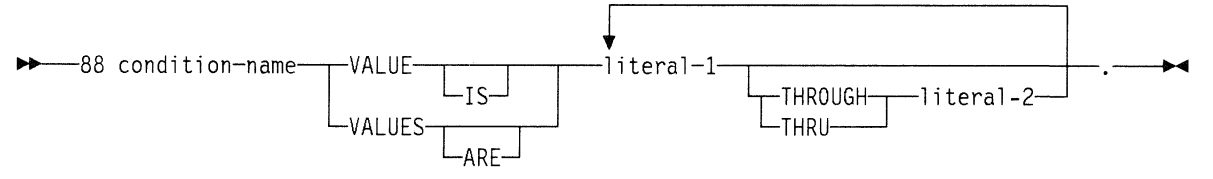
The VALUE clause must not be specified for any item whose length is variable.

If the VALUE clause is specified at the group level, the literal must be a nonnumeric literal or a figurative constant other than NULL or NULLS. The group area is initialized without consideration for the subordinate entries within this group. In addition, the VALUE clause must not be specified for subordinate entries within this group.

For group entries, the VALUE clause must not be specified if the entry also contains any of the following clauses: JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE DISPLAY).

The VALUE clause must not conflict with other clauses in the data description entry, or in the data description of this entry's hierarchy.

Format 2—Condition-Name Value



This format associates a value, values, and/or range(s) of values with a condition-name. Each such condition-name requires a separate level-88 entry. Level-number 88 and condition-name are not part of the Format 2 VALUE clause itself. They are included in the format only for clarity.

condition-name

A user-specified name that associates a value with a conditional variable. If the associated conditional variable requires subscripts or indexes, each procedural reference to the condition-name must be subscripted or indexed as required for the conditional variable.

Condition-names are tested procedurally in condition-name conditions (see "Conditional Expressions" on page 189).

literal-1

When literal-1 is specified alone, the condition-name is associated with a single value.

literal-1 THROUGH literal-2

The condition-name is associated with at least one range of values. Whenever the THROUGH phrase is used, literal-1 must be less than literal-2.

Rules for Condition-Name Values:

- The VALUE clause is required in a condition-name entry, and must be the only clause in the entry. Each condition-name entry is associated with a preceding conditional variable. Thus, every level-88 entry must always be preceded either by the entry for the conditional variable, or by another level-88 entry when several condition-names apply to one conditional variable. Each such level-88 entry implicitly has the PICTURE characteristics of the conditional variable.
- The key words THROUGH and THRU are equivalent.

  The condition-name entries associated with a particular conditional variable must immediately follow the conditional variable entry. The conditional variable can be any elementary data description entry except:

  - Another condition-name
  - A level-66 item
  - An index data item.

  The conditional variable cannot be an item whose usage is POINTER.

  A condition-name can be associated with a group item data description entry. In this case:

  - The condition-name value must be specified as a nonnumeric literal or figurative constant.
  - The size of the condition-name value must not exceed the sum of the sizes of all the elementary items within the group.
  - No element within the group may contain a JUSTIFIED or SYNCHRONIZED clause.
  - No USAGE other than DISPLAY may be specified within the group.

  Condition-names can be specified both at the group level and at subordinate levels within the group.

  The relation test implied by the definition of a condition-name at the group level is performed in accordance with the rules for comparison of nonnumeric operands, regardless of the nature of elementary items within the group.

  A space, a separator comma, or a separator semicolon, must separate successive operands.

  Each entry must end with a separator period.

- The type of literal in a condition-name entry must be consistent with the data type of its conditional variable. In the following example:
  - CITY-COUNTY-INFO, COUNTY-NO, and CITY are conditional variables.

    The PICTURE associated with COUNTY-NO limits the condition-name value to a 2-digit numeric literal.

    The PICTURE associated with CITY limits the condition-name value to a 3-character nonnumeric literal.
  - The associated condition-names are level-88 entries.

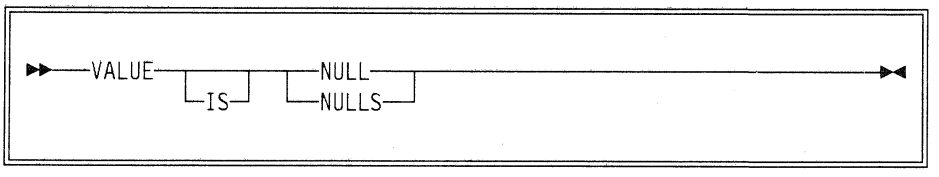
    Any values for the condition-names associated with CITY-COUNTY-INFO cannot exceed 5 characters.

    Because this is a group item, the literal must be nonnumeric.

```
05  CITY-COUNTY-INFO.
    88  BRONX                  VALUE "03NYC".
    88  BROOKLYN               VALUE "24NYC".
    88  MANHATTAN              VALUE "31NYC".
    88  QUEENS                 VALUE "41NYC".
    88  STATEN-ISLAND          VALUE "43NYC".
  10  COUNTY-NO                PICTURE 99.
      88  DUTCHESS             VALUE 14.
      88  KINGS                VALUE 24.
      88  NEW-YORK             VALUE 31.
      88  RICHMOND             VALUE 43.
  10  CITY                     PICTURE X(3).
      88  BUFFALO              VALUE "BUF".
      88  NEW-YORK-CITY        VALUE "NYC".
      88  POUGHKEEPSIE         VALUE "POK".
05  POPULATION...
```

IBM Extension

Format 3—NULL Value



This format assigns an address that is not valid to a pointer data item (an item whose usage is POINTER). A value of NULL is an undefined value.

VALUE IS NULL can only be specified for elementary items described implicitly or explicitly as USAGE IS POINTER.

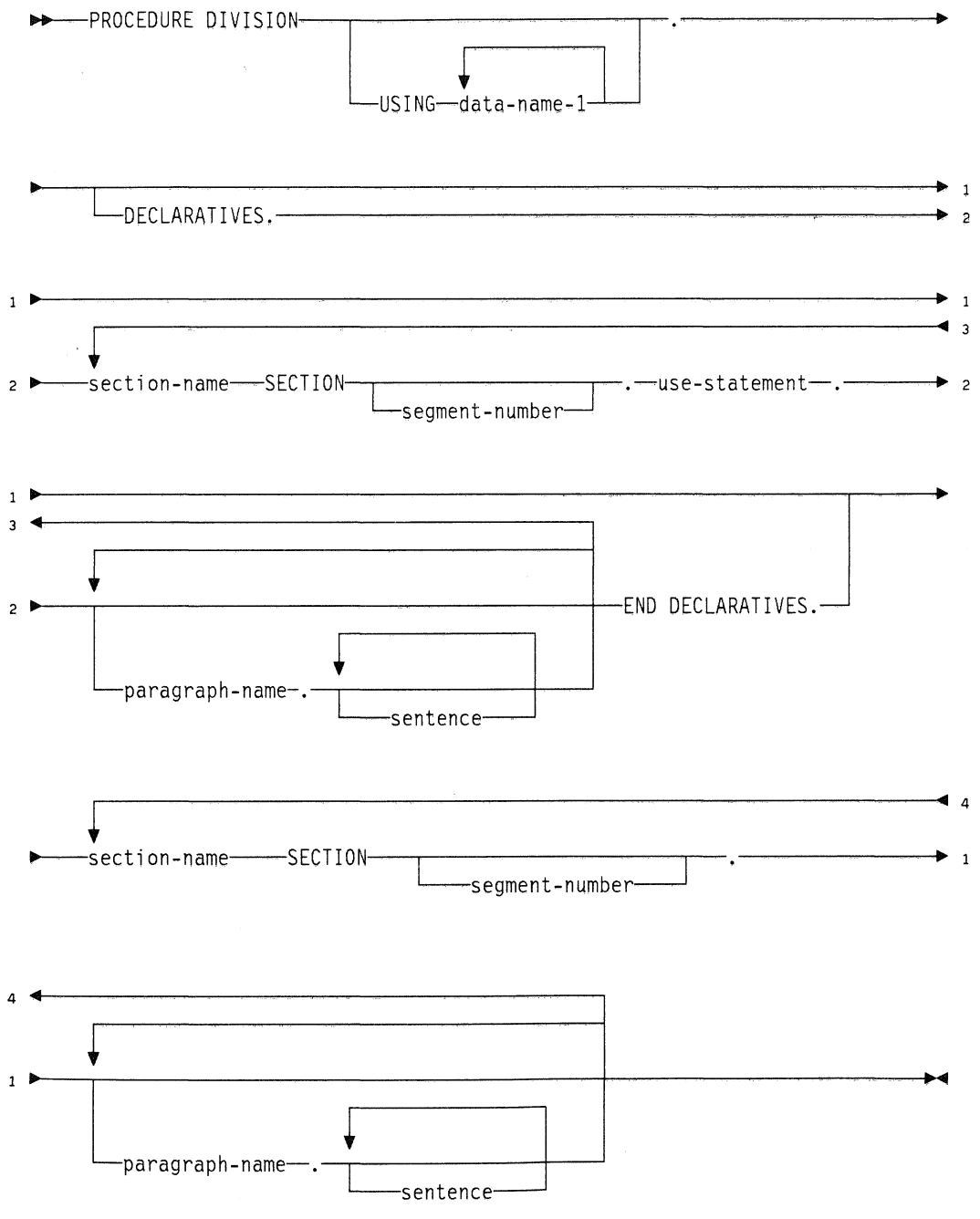
End of IBM Extension

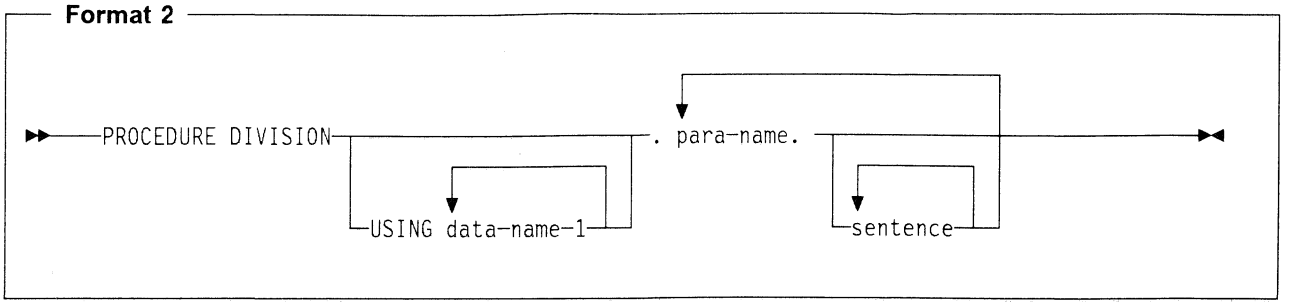
Procedure Division

The Procedure Division is optional in a COBOL source program. The Procedure Division consists of optional declaratives, and procedures that contain sections and/or paragraphs, sentences, and statements.

The structure of the Procedure Division is as follows:

Format 1





Coding Example

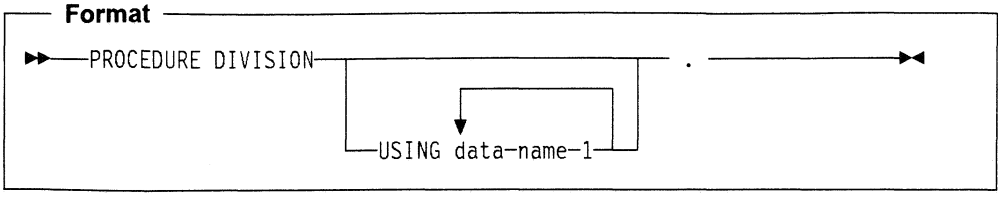
```
SEQUENCE      C  A       B
              O
PAGE SERIAL   N
              T
1    3 4    6 7 8       12      16      20      24      28      32      36
0 0 4 0 1 0   PROCEDURE  DIVISION.
      0 2 0   DECLARATIVES.
      0 3 0   SECTION-NAME  SECTION.
      0 4 0   PARAGRAPH-NAMES.
      0 5 0           PROGRAMMING  STATEMENTS.
      0 6 0 *         COMMENTS.
      0 7 0   END   DECLARATIVES.
      0 8 0   SECTION-NAME  SECTION.
      0 9 0   PARAGRAPH-NAME.
      1 0 0           PROGRAMMING  STATEMENTS.
```

Figure 13. Coding Example to Show Procedure Division Organization

The Procedure Division Header

The Procedure Division, if specified, is identified by the following header.

Format



The USING phrase is required only if the object program is to be invoked by a CALL statement and that statement includes a USING phrase.

The USING Phrase

The following discussion of the USING phrase assumes that the calling and called programs are written in COBOL.

The USING phrase makes data items defined in a calling program available to a called subprogram.

The USING phrase is specified in the Procedure Division header if, and only if, this program is a subprogram invoked by a CALL statement that itself contains a USING phrase. That is, for each CALL USING statement in a calling program, there must be a corresponding USING phrase specified in a called subprogram.

The USING phrase is valid in the Procedure Division header of a called subprogram entered at the beginning of the nondeclaratives portion; each USING identifier must be defined as a level-01 or level-77 item in the Linkage Section of the called subprogram; it must not contain a REDEFINES clause. A particular user-defined word may not appear more than once as data-name-1. In a calling program, the USING phrase is valid for the CALL statement; each USING identifier must be defined as a level-01, level-77, or an elementary item in the Data Division. The maximum number of data-names that can be specified is 30.

The order of appearance of USING identifiers in both calling and called subprograms determines the correspondence of single sets of data available to both programs. The correspondence is positional and not by name. Corresponding identifiers must contain the same number of characters, although their data descriptions need not be the same. For index-names, no correspondence is established; index-names in calling and called programs always refer to separate indexes.

The identifiers specified in a CALL USING statement name data items available to the calling program that may be referred to in the called program; a given identifier may appear more than once. These items are defined in any Data Division section.

IBM Extension

An identifier may appear more than once in a Procedure Division USING phrase. In that case, the last value assigned to the identifier by a CALL USING statement is used.

End of IBM Extension

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:

- They are operands of the USING phrase of the Procedure Division header
- They are defined with a REDEFINES or RENAMES clause, the object of which satisfies the above condition

IBM Extension

- They are used as arguments of the ADDRESS OF special register

End of IBM Extension

- They are items subordinate to any item which satisfies the condition in the rules above
- They are condition-names or index-names associated with data items that satisfy any of the above conditions.

Declaratives

Declaratives provide one or more special-purpose sections that are executed when an exceptional condition occurs.

When Declarative Sections are specified, they must be grouped at the beginning of the Procedure Division, and the entire Procedure Division must be divided into sections.

Each Declarative Section starts with a USE sentence that identifies the section's function; the series of procedures that follow specify what actions are to be taken when the exceptional condition occurs. Each Declarative Section ends with another section-name followed by a USE sentence, or with the key words END DECLARATIVES. See "USE Statement" on page 475 for more information on the USE statement.

The entire group of Declarative Sections is preceded by the key word DECLARATIVES, written on the line after the Procedure Division header; the group is followed by the key words END DECLARATIVES. The key words DECLARATIVES and END DECLARATIVES must each begin in Area A and be followed by a separator period. No other text may appear on the same line.

In the declaratives part of the Procedure Division, each section header (with an optional segment number) must be followed by a separator period, and must be followed by a USE sentence, followed by a separator period. No other text may appear on the same line.

The USE sentence itself is never executed; instead, the USE sentence defines the conditions that execute the succeeding procedural paragraphs, which specify the actions to be taken. After the procedure is executed, control is returned to the routine that activated it.

Within a declarative procedure, except for the USE statement itself, there must be no reference to any nondeclarative procedure.

Within a declarative procedure, no statement should be included that would cause the execution of a USE procedure that had been previously invoked and had not yet returned control to the invoking routine.

The declarative procedure is exited when the last statement in the procedure is executed.

Procedures

Within the Procedure Division, a **procedure** consists of:

- A **section** or a group of sections
- A **paragraph** or group of paragraphs.

A **procedure-name** is a user-defined name that identifies a section or a paragraph.

Section
A **section header** optionally followed by one or more paragraphs.

Section-header
A **section-name** followed by:

the keyword SECTION
an optional segment-number
a separator period.

The section-header must begin in Area A. Segment-numbers are explained in the *COBOL/400 User's Guide*.

Section-name
A user-defined word that identifies a section. If referenced, a section-name must be unique within the program in which it is defined, because it cannot be qualified.

A section ends immediately before the next section header, or at the end of the Procedure Division, or, in the declaratives portion, at the key words END DECLARATIVES.

Paragraph
A **paragraph-name** followed by a separator period, optionally followed by one or more sentences.

Paragraph-name
A user-defined word that identifies a paragraph. A paragraph-name, because it can be qualified, need not be unique. The paragraph-name must begin in Area A.

A paragraph ends immediately before the next paragraph-name or section header, or at the end of the Procedure Division, or, in the declaratives portion, at the key words END DECLARATIVES.

If one paragraph in a program is contained within a section, all paragraphs must be contained in sections.

Sentence
One or more **statements** terminated by a separator period.

Statement
A syntactically valid combination of **identifiers** and symbols (literals, relational-operators, and so forth) beginning with a COBOL verb.

identifier
A syntactically correct combination of a data-name, with its qualifiers, subscripts, and reference modifiers as required for uniqueness of reference, that names a data item. In any Procedure Division reference (except the class test), the contents of an identifier must be compatible with the class specified through its PICTURE clause, or results are unpredictable.

Execution begins with the first statement in the Procedure Division, excluding declaratives. Statements are executed in the order in which they are presented for compilation, unless the statement rules dictate some other order of execution.

The Procedure Division ends at the physical end of the program; that is, the physical position in a source program after which no further statements appear.

Sample Procedure Division Statements

```
. 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7

PROCEDURE DIVISION.
DECLARATIVES.
ERROR-IT SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON INPUT-DATA.
ERROR-ROUTINE.
    IF CHECK-IT = "30" ADD 1 TO DECLARATIVE-ERRORS.
END DECLARATIVES.
BEGIN-NON-DECLARATIVES SECTION.
100-BEGIN-IT.
    OPEN INPUT INPUT-DATA OUTPUT REPORT-OUT.
110-READ-IT.
    READ INPUT-DATA RECORD
       AT END MOVE "Y" TO EOF-SW.
    IF EOF-SW NOT = "Y" ADD 1 TO RECORDS-IN.
200-MAIN-ROUTINE.
    PERFORM PROCESS-DATA UNTIL EOF-SW = "Y".
    PERFORM FINAL-REPORT THRU FINAL-REPORT-EXIT.
    DISPLAY "TOTAL RECORDS IN = " RECORDS-IN UPON WORK-STATION.
    DISPLAY "DECLARATIVE ERRORS = " DECLARATIVE-ERRORS
       UPON WORK-STATION.
    STOP RUN.
PROCESS-DATA.
    IF RECORD-ID = "G"
        PERFORM PROCESS-GEN-INFO
    ELSE
        IF RECORD-CODE = "C"
           PERFORM PROCESS-SALES-DATA
        ELSE
           PERFORM UNKNOWN-RECORD-TYPE.
```

Arithmetic Expressions

Arithmetic expressions are used as operands of certain conditional and arithmetic statements.

An arithmetic expression may consist of any of the following:

1. An identifier described as a numeric elementary item
2. A numeric literal
3. The figurative constant ZERO
4. Identifiers and literals, as defined in items 1, 2, and 3, separated by arithmetic operators
5. Two arithmetic expressions, as defined in items 1, 2, 3, and/or 4, separated by an arithmetic operator
6. An arithmetic expression, as defined in items 1, 2, 3, 4, and/or 5, enclosed in parentheses.

Any arithmetic expression may be preceded by a unary operator.

Identifiers and literals appearing in arithmetic expressions must represent either numeric elementary items or numeric literals on which arithmetic may be performed.

If an exponential expression is evaluated as both a positive and a negative number, the result will always be the positive number. The square root of 4, for example,

```
4 ** 0.5 (the square root of 4)
```

is evaluated as +2 and -2. The COBOL/400 result is always +2.

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. In any case where no real number exists as the result of the evaluation, the size error condition exists.

Arithmetic Operators

Five binary arithmetic operators and two unary arithmetic operators (Table 14) may be used in arithmetic expressions. They are represented by specific characters that must be preceded and followed by a space.

Table 14. Binary and Unary Operators

Binary Operator	Meaning	Unary Operator	Meaning
+	Addition	+	Multiplication by +1
–	Subtraction	–	Multiplication by –1
*	Multiplication		
/	Division		
**	Exponentiation		

Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated.

Expressions within parentheses are evaluated first. When expressions are contained within a nest of parentheses, evaluation proceeds from the innermost to the outermost set.

When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchic order is implied:

1. Unary operator
2. Exponentiation
3. Multiplication and division
4. Addition and subtraction.

Parentheses either eliminate ambiguities in logic where consecutive operations appear at the same hierarchic level or modify the normal hierarchic sequence of execution when this is necessary. When the order of consecutive operations at the same hierarchic level is not completely specified by parentheses, the order is from left to right.

An arithmetic expression may begin only with a left parenthesis, a unary operator, or an operand (that is, an identifier or a literal). It may end only with a right parenthesis or an operand. An arithmetic expression must contain at least one reference to an identifier or a literal.

There must be a one-to-one correspondence between left and right parentheses in an arithmetic expression, with each left parenthesis placed 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.

Table 15 shows permissible arithmetic symbol pairs. An arithmetic symbol pair is the combination of two such symbols in sequence. In the figure:

Yes indicates a permissible pairing.
No indicates that the pairing is not permitted.

Table 15. Valid Arithmetic Symbol Pairs

	Second Symbol				
First Symbol	**Identifier or Literal**	*** / ** + –**	**Unary + or Unary –**	**(**	**)**
Identifier or Literal	No	Yes	No	No	Yes
*** / ** + –**	Yes	No	Yes	Yes	No
Unary + or Unary –	Yes	No	No	Yes	No
(	Yes	No	Yes	Yes	No
)	No	Yes	No	No	Yes

Programming Notes: The results of exponentiation are truncated after the thirteenth fractional digit. The results of exponentiation when the exponent is noninteger are accurate to seven digits.

Conditional Expressions

A conditional expression causes the object program to select alternative paths of control, depending on the truth value of a test. Conditional expressions are specified in EVALUATE, IF, PERFORM, and SEARCH statements.

A conditional expression can be specified in either simple conditions or complex conditions. Both simple and complex conditions can be enclosed within any number of paired parentheses; the parentheses do not change whether the condition is simple or complex.

Simple Conditions

There are five simple conditions:

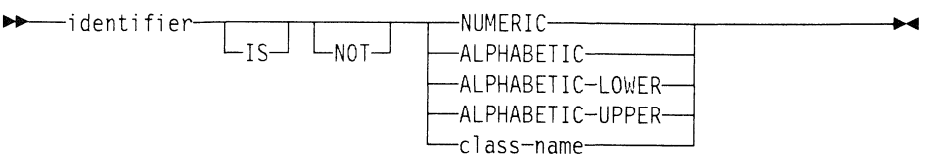
- Class condition
- Condition-name condition
- Relation condition
- Sign condition
- Switch-status condition.

A simple condition has a truth value of either true or false.

Class Condition

The class condition determines whether the content of a data item is alphabetic, alphabetic-lower, alphabetic-upper, numeric, 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.

Format



NOT

When used, NOT and the next key word define the class test to be executed for truth value. For example, NOT NUMERIC is a truth test for determining that an identifier is nonnumeric.

NUMERIC

The data item consists entirely of the characters 0 through 9, with or without an operational sign.

If its PICTURE does not contain an operational sign, the item being tested is determined to be numeric only if the contents are numeric and an operational sign is not present.

If its PICTURE does contain an operational sign, the item being tested is determined to be numeric only if the item is an elementary item, the contents are numeric, and a valid operational sign is present.

In the EBCDIC collating sequence, valid embedded operational positive signs are hexadecimal F, C, E, and A. Negative signs are hexadecimal D and B. The preferred positive sign is hexadecimal F, and the preferred negative sign is hexadecimal D. For items described with the SIGN IS SEPARATE clause, valid operational signs are + (hex 4E) and – (hex 60).

The NUMERIC test cannot be used with an identifier described as alphabetic or as a group item that contains one or more signed elementary items.

IBM Extension

For numeric data items, the identifier being tested can be described implicitly or explicitly as USAGE DISPLAY, USAGE PACKED-DECIMAL, USAGE COMP, or USAGE COMP-3.

End of IBM Extension

ALPHABETIC

The data item consists entirely of any combination of the lowercase or uppercase alphabetic characters A through Z, and the space.

The ALPHABETIC test cannot be used with an identifier described as numeric.

ALPHABETIC-LOWER
The data item consists entirely of any combination of the lowercase alphabetic characters a through z, and the space.

The ALPHABETIC-LOWER test cannot be used with an identifier described as numeric.

ALPHABETIC-UPPER
The data item consists entirely of any combination of the uppercase alphabetic characters A through Z, and the space.

The ALPHABETIC-UPPER test cannot be used with an identifier described as numeric.

class-name
The data item consists entirely of the characters listed in the definition of class-name in the SPECIAL-NAMES paragraph.

The class-name test must not be used with an identifier described as numeric.

The class test is not valid for items whose usage is INDEX or POINTER because these items do not belong to any class or category.

Table 16 shows valid forms of the class test.

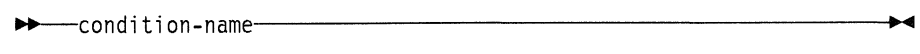
Table 16. Valid Forms of the Class Test

Type of Identifier	Valid Forms of the Class Test	
Alphabetic	ALPHABETIC ALPHABETIC-LOWER ALPHABETIC-UPPER class-name	NOT ALPHABETIC NOT ALPHABETIC-LOWER NOT ALPHABETIC-UPPER NOT class-name
Alphanumeric, Alphanumeric-edited, or Numeric-edited	ALPHABETIC ALPHABETIC-LOWER ALPHABETIC-UPPER NUMERIC class-name	NOT ALPHABETIC NOT ALPHABETIC-LOWER NOT ALPHABETIC-UPPER NOT NUMERIC NOT class-name
External-Decimal	NUMERIC	NOT NUMERIC

Condition-Name Condition

A condition-name condition tests a conditional variable to determine whether its value is equal to any value(s) associated with the condition-name.

Format



A condition-name is used in conditions as an abbreviation for the relation condition. The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

If the condition-name has been associated with a range of values (or with several ranges of values), the conditional variable is tested to determine whether or not its value falls within the range(s), including the end values. The result of

the test is true if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

The following example illustrates the use of conditional variables and condition-names:

```
01  AGE-GROUP          PIC  99.
    88  INFANT         VALUE 0.
    88  BABY           VALUE 1, 2.
    88  CHILD          VALUE 3 THRU 12.
    88  TEEN-AGER      VALUE 13 THRU 19.
```

AGE-GROUP is the conditional variable; INFANT, BABY, CHILD, and TEEN-AGER are condition-names. For individual records in the file, only one of the values specified in the condition-name entries can be present.

The following IF statements can be added to the above example to determine the age group of a specific record:

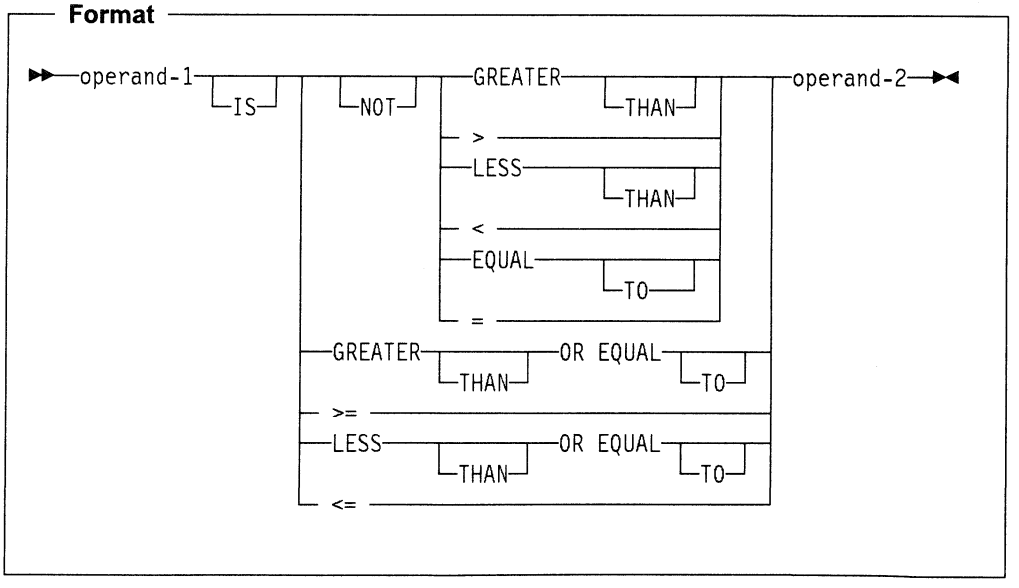
```
IF INFANT...            (Tests for value 0)
IF BABY...              (Tests for values 1, 2)
IF CHILD...             (Tests for values 3 through 12)
IF TEEN-AGER...         (Tests for values 13 through 19)
```

Depending on the evaluation of the condition-name condition, alternative paths of execution are taken by the object program.

Relation Condition

A relation condition compares two operands, either of which may be an identifier, a literal, an arithmetic expression, or an index-name.

Format



operand-1
The subject of the relation condition.

operand-2
The object of the relation condition.

Operand-1 and operand-2 may each be an identifier, a literal, an arithmetic expression, or an index-name. The relation condition must contain at least one reference to an identifier.

The relational operator specifies the type of comparison to be made. Table 17 shows relational operators and their meanings. Each relational operator must be preceded and followed by a space.

Table 17. Relational Operators and Their Meanings

Relational Operator	Can Be Written	Meaning
IS GREATER THAN	IS >	Greater than
IS NOT GREATER THAN	IS NOT >	Not greater than
IS LESS THAN	IS <	Less than
IS NOT LESS THAN	IS NOT <	Not less than
IS EQUAL TO	IS =	Equal to
IS NOT EQUAL TO	IS NOT =	Not equal to
IS GREATER THAN OR EQUAL TO	IS > =	Is greater than or equal to
IS LESS THAN OR EQUAL TO	IS < =	Is less than or equal to

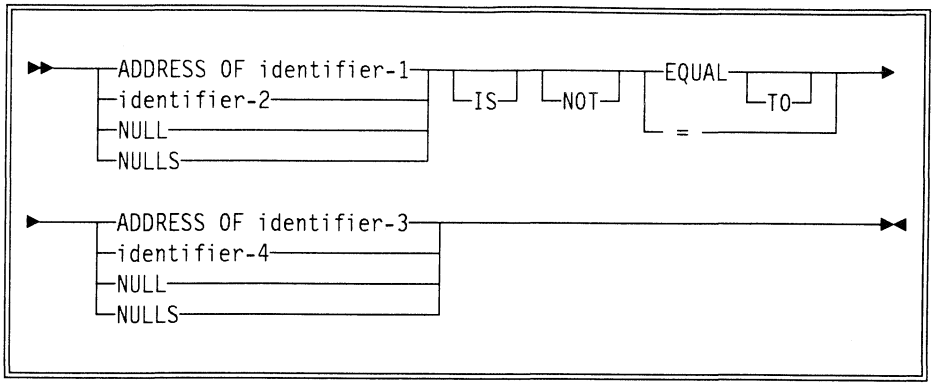
IBM Extension

Pointer Data Items: Only EQUAL and NOT EQUAL are allowed as relational operators when you specify pointer data items. Pointer data items are items defined explicitly as USAGE IS POINTER. Otherwise, they are ADDRESS OF data items or ADDRESS OF special registers, which are implicitly defined as USAGE IS POINTER.

The operands are equal if the two addresses used in the comparison would both result in the same storage location.

This relation condition is allowed in IF, PERFORM, EVALUATE, and SEARCH Format 1 statements. It is not allowed in SEARCH Format 2 (SEARCH ALL) statements, because there is not a meaningful ordering that can be applied to pointer data items.

Format-ADDRESS Comparison



identifier-1
identifier-3
May specify any level item defined in the Linkage Section, except 66 and 88.

identifier-2
identifier-4
Must be described as USAGE IS POINTER.

NULL(S)
Can be used only if the other operand is one of these:

- An item whose usage is POINTER
- The ADDRESS OF an item
- The ADDRESS OF special register.

That is, NULL=NULL is not allowed.

End of IBM Extension

Rules for numeric and nonnumeric comparisons are given in the following tables. If either of the operands is a group item, nonnumeric comparison rules apply.

Table 18 on page 195 summarizes permissible comparisons with *nonnumeric* operands.

Table 19 on page 196 summarizes permissible comparisons with *numeric* operands.

The symbols used in Table 18 and Table 19 are as follows:

NN = Comparison for nonnumeric operands.
NU = Comparison for numeric operands.
Blank = Comparison is not allowed.

Table 18. Permissible Comparisons with Nonnumeric Second Operands

FIRST OPERAND	SECOND OPERAND						
	GR	AL	AN	ANE	NE	FC	NNL
NONNUMERIC OPERAND							
Group (GR)	NN	NN	NN	NN	NN	NN[1]	NN
Alphabetic (AL)	NN	NN	NN	NN	NN	NN[1]	NN
Alphanumeric (AN)	NN	NN	NN	NN	NN	NN[1]	NN
Alphanumeric Edited (ANE)	NN	NN	NN	NN	NN	NN[1]	NN
Numeric Edited (NE)	NN	NN	NN	NN	NN	NN[1]	NN
Figurative Constant (FC[1])	NN	NN	NN	NN	NN		
Nonnumeric Literal (NNL)	NN	NN	NN	NN	NN		
NUMERIC OPERAND							
Figurative Constant ZERO (ZR)	NN	NN	NN	NN			
Numeric Literal (NL)	NN	NN	NN	NN			
External Decimal (ED)	NN[2]	NN[2]	NN[2]	NN[2]	NN[2]	NN[1 2]	NN[1]
Binary (BI)							
Arithmetic Expression (AE)							
IBM Extension Boolean data item or Boolean literal (BO) End of IBM Extension							
Internal Decimal (ID)							

Table 19. Permissible Comparisons with Numeric Second Operands

FIRST OPERAND	SECOND OPERAND						
	ZR	NL	ED	BI	AE	BO	ID
NONNUMERIC OPERAND							
Group (GR)	NN	NN	NN[2]				
Alphabetic (AL)	NN	NN	NN[2]				
Alphanumeric (AN)	NN	NN	NN[2]				
Alphanumeric Edited (ANE)	NN	NN	NN[2]				
Numeric Edited (NE)	NN	NN	NN[2]				
Figurative Constant (FC[1])			NN[2]				
Nonnumeric Literal (NNL)			NN[2]				
NUMERIC OPERAND							
Figurative Constant ZERO (ZR)			NU	NU	NU	NU	NU
Numeric Literal (NL)			NU	NU	NU		NU
External Decimal (ED)	NU	NU	NU	NU	NU		NU
Binary (BI)	NU	NU	NU	NU	NU		NU
Arithmetic Expression (AE)	NU	NU	NU	NU	NU		NU
IBM Extension Boolean data item or Boolean literal (BO) End of IBM Extension						NU	
Internal Decimal (ID)	NU	NU	NU	NU	NU		NU

Notes to Table 18 on page 195 and Table 19:

1 Includes all figurative constants except ZERO and NULL
2 Integer item only

Comparison of Numeric and Nonnumeric Operands

Comparing Numeric Operands: The algebraic values of numeric operands are compared.

- The length (number of digits) of the operands is not significant.
- Unsigned numeric operands are considered positive.
- Zero is considered to be a unique value, regardless of sign.
- Comparison of numeric operands is permitted, regardless of the type of USAGE specified for each.

Comparing Nonnumeric Operands: Comparisons of nonnumeric operands are made with respect to the collating sequence of the character set in use.

When the PROGRAM COLLATING SEQUENCE clause is specified in the OBJECT-COMPUTER paragraph, the collating sequence associated with the alphabet-name clause in the SPECIAL-NAMES paragraph is used. Otherwise, the native EBCDIC character set is used.

The size of each operand is the total number of characters in that operand; the size affects the result of the comparison. There are two cases to consider:

Operands of Equal Size
Characters in corresponding positions of the two operands are compared, beginning with the leftmost character and continuing through the rightmost character.

If all pairs of characters through the last pair test as equal, the operands are considered as equal.

If a pair of unequal characters is encountered, the characters are tested to determine their relative positions in the collating sequence. The operand containing the character higher in the sequence is considered the greater operand.

Operands of Unequal Size
If the operands are of unequal size, the comparison is made as though the shorter operand were extended to the right with enough spaces to make the operands equal in size.

Comparing Numeric and Nonnumeric Operands: The nonnumeric comparison rules, discussed above, apply. In addition, when numeric and nonnumeric operands are being compared, their USAGE must be the same. In such comparisons:

- The numeric operand must be described as an integer literal or data item.
- Noninteger literals and data items must not be compared with nonnumeric operands.

If either of the operands is a group item, the nonnumeric comparison rules, discussed above, apply. In addition to those rules:

- If the nonnumeric operand is a **literal or an elementary data item**, the numeric operand is treated as though it were moved to an alphanumeric elementary data item of the same size, and the contents of this alphanumeric data item were then compared with the nonnumeric operand.
- If the nonnumeric operand is a **group item**, the numeric operand is treated as though it were moved to a group item of the same size, and the contents of this group item were compared then with the nonnumeric operand.

(See "MOVE Statement" on page 321.)

IBM Extension

Comparison of Boolean Operands: Boolean operands can be used only in the [NOT] EQUAL TO relation condition. Boolean operands cannot be compared to non-Boolean operands. Boolean data items and literals must be one position in length. Two Boolean operands are equal if they both have a value of Boolean 1 or Boolean 0. The Boolean operands are unequal if one has a value of Boolean 1 and the other has a value of Boolean 0.

End of IBM Extension

Comparing Index-Names and Index Data Items: Comparisons involving index-names and/or index data items conform to the following rules:

- The comparison of two index-names is actually the comparison of the corresponding occurrence numbers.
- In the comparison of an index-name with a data item (other than an index data item), or in the comparison of an index-name with a literal, the occurrence number that corresponds to the value of the index-name is compared with the data item or literal.
- In the comparison of an index data item with an index-name or another index data item, the actual values are compared without conversion. Results of any other comparison involving an index data item are undefined.

Table 20 shows valid comparisons involving index-names and index data items.

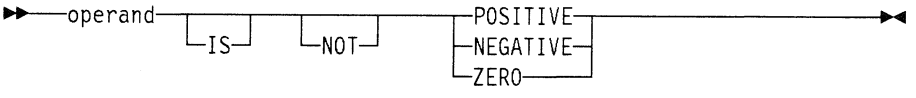
Table 20. Comparisons Involving Index Names and Index Data Items

Operands Compared	**Index-Name**	**Index Data Item**	**Data-Name**	**Literal**
Index-Name	Compare occurrence number	Compare without conversion	Compare occurrence number with data-name	Compare occurrence number with literal
Index Data Item	Compare without conversion	Compare without conversion	Not permitted	Not permitted

Sign Condition

The sign condition determines whether or not the algebraic value of a numeric operand is greater than, less than, or equal to zero.

Format



operand

Must be defined as a numeric identifier, or it must be defined as an arithmetic expression that contains at least one reference to an identifier.

The operand is:

- POSITIVE if its value is greater than zero
- NEGATIVE if its value is less than zero
- ZERO if its value is equal to zero.

An unsigned operand is either POSITIVE or ZERO.

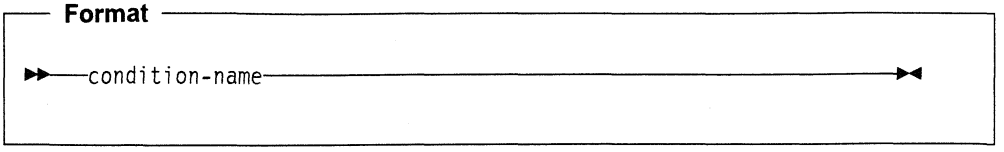
NOT

An algebraic test is executed for the truth value of the sign condition. For example, NOT ZERO is regarded as true when the operand tested is positive or negative in value.

Switch-Status Condition

The switch-status condition determines the on or off status of an UPSI switch.

Format



condition-name
Must be defined in the SPECIAL-NAMES paragraph as associated with the ON or OFF value of an UPSI switch. (See "SPECIAL-NAMES Paragraph" on page 57.)

The switch-status condition tests the value associated with the condition-name. (The value associated with the condition-name is considered to be alphanumeric.) The result of the test is true if the UPSI switch is set to the value (0 or 1) corresponding to condition-name.

Complex Conditions

A complex condition is formed by combining simple conditions, combined conditions, and/or complex conditions with logical operators, or negating these conditions with logical negation.

Each logical operator must be preceded and followed by a space. The following chart shows the logical operators and their meanings.

Table 21. Logical Operators and Their Meanings

Logical Operator	Name	Meaning
AND	Logical conjunction	The truth value is **true** when both conditions are true.
OR	Logical inclusive OR	The truth value is **true** when either or both conditions are true.
NOT	Logical negation	Reversal of truth value (the truth value is **true** if the condition is false).

Unless modified by parentheses, the following precedence rules (from highest to lowest) apply:

1. Arithmetic operations
2. Simple conditions
3. NOT
4. AND
5. OR

The truth value of a complex condition (whether parenthesized or not) is the truth value that results from the interaction of all the stated logical operators on either of the following:

- The individual truth values of simple conditions
- The intermediate truth values of conditions logically combined or logically negated.

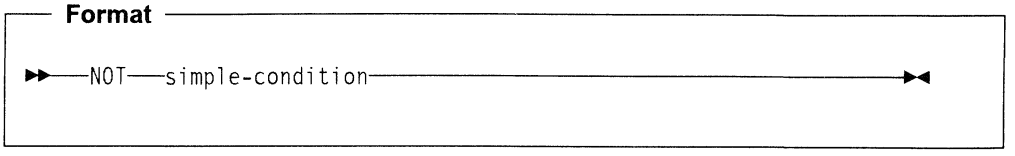
A complex condition can be either of the following:

- A negated simple condition
- A combined condition (which can be negated).

Negated Simple Conditions

A simple condition is negated through the use of the logical operator NOT.

Format



The negated simple condition gives the opposite truth value of the simple condition. That is, if the truth value of the simple condition is true, then the truth value of that same negated simple condition is false, and vice versa.

Placing a negated simple condition within parentheses does not change its truth value. That is, the following two statements are equivalent:

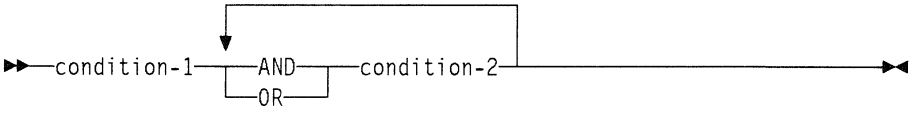
```
NOT A IS EQUAL TO B.

NOT (A IS EQUAL TO B).
```

Combined Conditions

Two or more conditions can be logically connected to form a combined condition.

Format



The condition to be combined may be any of the following:

- A simple-condition
- A negated simple-condition
- A combined condition
- A negated combined condition (that is, the NOT logical operator followed by a combined condition enclosed in parentheses)
- Combinations of the preceding conditions, specified according to the rules in Figure 14 on page 201.

Combined condition element	Leftmost	When not leftmost, may be immediately preceded by:	Rightmost	When not rightmost, may be immediately followed by:
simple-condition	Yes	OR NOT AND (	Yes	OR AND)
OR AND	No	simple-condition)	No	simple-condition NOT (
NOT	Yes	OR AND (	No	simple-condition (
(	Yes	OR NOT AND (	No	simple-condition NOT (
)	No	simple-condition)	Yes	OR AND)

Figure 14. Combined Conditions—Permissible Element Sequences

Parentheses are never needed when either ANDs or ORs (but not both) are used exclusively in one combined condition. However, parentheses may be needed to modify the implicit precedence rules to maintain the correct logical relation of operators and operands.

There must be a one-to-one correspondence between left and right parentheses, with each left parenthesis to the left of its corresponding right parenthesis.

Figure 15 illustrates the relationships between logical operators and conditions C1 and C2.

Value For C1	Value For C2	C1 AND C2	C1 OR C2	NOT (C1 AND C2)	NOT C1 AND C2	NOT (C1 OR C2)	NOT C1 OR C2
True	True	True	True	False	False	False	True
False	True	False	True	True	True	False	True
True	False	False	True	True	False	False	False
False	False	False	False	True	False	True	True

Figure 15. Logical Operators and Evaluation Results of Combined Conditions

Evaluating Conditional Expressions: If parentheses are used, logical evaluation of combined conditions proceeds in the following order:

1. Conditions within parentheses are evaluated first.
2. Within nested parentheses, evaluation proceeds from the least inclusive condition to the most inclusive condition.

If parentheses are not used (or are not at the same level of inclusiveness), the combined condition is evaluated in the following order:

1. Arithmetic expressions.
2. Simple-conditions in the following order:
   a. Relation
   b. Class
   c. Condition-name
   d. Switch-status
   e. Sign.
3. Negated simple-conditions in the same order as item 2.
4. Combined conditions, in the following order:
   a. AND
   b. OR.
5. Negated combined conditions in the following order:
   a. AND
   b. OR.
6. Consecutive operands at the same evaluation-order level are evaluated from left to right. However, the truth value of a combined condition can sometimes be determined without evaluating the truth value of all the component conditions.

The component conditions of a combined condition are evaluated from left to right. If the truth value of one condition is not affected by the evaluation of further elements of the combined condition, these elements are not evaluated. However, the truth value of the condition will always be the same (as if the condition had been evaluated in full), as described earlier in this paragraph.

For example:

```
NOT A IS GREATER THAN B OR A + B IS EQUAL
  TO C AND D IS POSITIVE
```

is evaluated as if parenthesized as follows:

```
(NOT (A IS GREATER THAN B)) OR (((A+B) IS EQUAL
  TO C) AND (D IS POSITIVE))
```

The order of evaluation in this example is as follows:

1. (NOT (A IS GREATER THAN B)) is evaluated. If true, the rest of the condition is not evaluated, as the expression is true.
2. (A+B) is evaluated, giving some intermediate result, x.
3. (x IS EQUAL TO C) is evaluated. If false, the rest of the condition is not evaluated, as the expression is false.

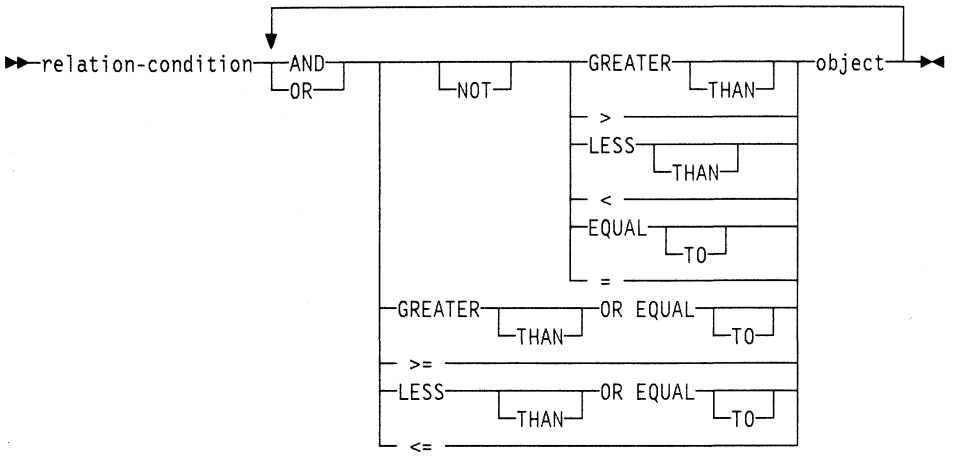
4. (D IS POSITIVE) is evaluated, giving the final truth value of the expression.

Abbreviated Combined Relation Conditions

When relation-conditions are written consecutively, any relation-condition after the first can be abbreviated in one of two ways:

- Omission of the subject
- Omission of the subject and relational operator.

Format



In any consecutive sequence of relation-conditions, both forms of abbreviation can be specified. The abbreviated condition is evaluated as if:

1. The last stated subject is the missing subject.
2. The last stated relational operator is the missing relational operator.

The resulting combined condition must comply with the rules for element sequence in combined conditions, as shown in Figure 14 on page 201.

The word NOT is considered part of the relational operator in the forms NOT GREATER THAN, NOT >, NOT LESS THAN, NOT <, NOT EQUAL TO, and NOT =.

NOT in any other position is considered a logical operator (and thus results in a negated relation-condition).

The following examples illustrate abbreviated combined relation conditions, with and without parentheses, and their unabbreviated equivalents.

Table 22. Abbreviated Combined Relation Conditions

Abbreviated Combined Relation Condition	Equivalent
A = B AND NOT < C OR D	((A = B) AND (A NOT < C)) OR (A NOT < D)
A NOT > B OR C	(A NOT > B) OR (A NOT > C)
NOT A = B OR C	(NOT (A = B)) OR (A = C)
NOT (A = B OR < C)	NOT ((A = 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))))

Statement Categories

There are four categories of COBOL statements:

- Imperative
- Conditional
- Delimited scope
- Compiler directing.

Imperative Statements

An **imperative statement** either specifies an unconditional action to be taken by the program, or is a conditional statement terminated by its explicit scope terminator (see "Delimited Scope Statements" on page 207). A series of imperative statements can be specified whenever an imperative statement is allowed.

Table 23 lists COBOL imperative statements.

Table 23. Types of Imperative Statements

Type	Imperative Statement
Arithmetic	ADD[1] COMPUTE[1] DIVIDE[1] INSPECT (TALLYING) MULTIPLY[1] SUBTRACT[1]
Data Movement	ACCEPT (DATE, DAY, TIME) INITIALIZE INSPECT (CONVERTING) INSPECT (REPLACING) MOVE STRING[2] UNSTRING[2]
Ending	STOP RUN EXIT PROGRAM IBM Extension GOBACK End of IBM Extension
Input/Output	ACCEPT[6] CLOSE DELETE[3] DISPLAY OPEN READ[4] REWRITE[3] START[3] STOP literal WRITE[5] IBM Extension ACQUIRE COMMIT DROP ROLLBACK End of IBM Extension
Ordering	MERGE RELEASE RETURN SORT
Procedure Branching	ALTER EXIT GO TO PERFORM
Subprogram Linkage	CALL[7] CANCEL
Table Handling	SET

Notes to Table 23:

1. Without the ON SIZE ERROR or NOT ON SIZE ERROR phrase.
2. Without the NOT ON OVERFLOW or ON OVERFLOW phrase.
3. Without the INVALID KEY or NOT INVALID KEY phrase.
4. Without the AT END, NOT AT END, INVALID KEY, NO DATA, or NOT INVALID KEY phrase.
5. Without the INVALID KEY, NOT INVALID KEY, END-OF-PAGE, or NOT END-OF-PAGE phrase.
6. Without the ON EXCEPTION or NOT ON EXCEPTION phrase.
7. Without the ON OVERFLOW, ON EXCEPTION, or NOT ON EXCEPTION phrase.

Conditional Statements

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. (See "Conditional Expressions" on page 189.)

Figure 16 lists COBOL statements that are conditional, or that become conditional when a **condition** is included and the statement is not terminated by its explicit scope terminator.

Arithmetic

ADD...ON SIZE ERROR
ADD...NOT ON SIZE ERROR
COMPUTE...ON SIZE ERROR
COMPUTE...NOT ON SIZE ERROR
DIVIDE...ON SIZE ERROR
DIVIDE...NOT ON SIZE ERROR
MULTIPLY...ON SIZE ERROR
MULTIPLY...NOT ON SIZE ERROR
SUBTRACT...ON SIZE ERROR
SUBTRACT...NOT ON SIZE ERROR

Ordering

RETURN...AT END
RETURN...NOT AT END

Data Movement

STRING...ON OVERFLOW
STRING...NOT ON OVERFLOW
UNSTRING...ON OVERFLOW
UNSTRING...NOT ON OVERFLOW

Subprogram Linkage

CALL...ON OVERFLOW
CALL...ON EXCEPTION
CALL...NOT ON EXCEPTION

Decision

IF
EVALUATE

Table Handling

SEARCH...WHEN

Input/Output

ACCEPT...ON EXCEPTION
ACCEPT...NOT ON EXCEPTION
DELETE...INVALID KEY
DELETE...NOT INVALID KEY
READ...AT END
READ...NOT AT END
READ...INVALID KEY
READ...NOT INVALID KEY
READ...NO DATA
REWRITE...INVALID KEY
REWRITE...NOT INVALID KEY
START...INVALID KEY
START...NOT INVALID KEY
WRITE...AT END-OF-PAGE
WRITE...NOT AT END-OF-PAGE
WRITE...INVALID KEY
WRITE...NOT INVALID KEY

Figure 16. Conditional Statements

Delimited Scope Statements

In general, a delimited scope statement uses an explicit scope terminator to turn a conditional statement into an imperative statement; the resulting imperative statement can then be nested. Explicit scope terminators may also be used, however, to terminate the scope of an imperative statement. Explicit scope terminators are provided for all COBOL verbs that may have conditional phrases.

Unless explicitly specified otherwise, a delimited scope statement may be specified wherever an imperative statement is allowed by the rules of the language.

Explicit Scope Terminators

An explicit scope terminator marks the end of certain Procedure Division statements. A conditional statement that is delimited by its explicit scope terminator is considered an imperative statement and must follow the rules for imperative statements.

The following are explicit scope terminators:

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

Implicit Scope Terminators

At the end of any sentence, an implicit scope terminator is a separator period that terminates the scope of all previous statements not yet terminated.

A conditional statement not terminated by its scope terminator cannot be contained within another statement.

Note: Except for nesting conditional statements within IF statements, nested statements must be imperative statements, and must follow the rules for imperative statements. You should not nest conditional statements.

Compiler-Directing Statements

Statements that direct the compiler to take a specified action are discussed in "Part 4. Compiler-Directing Statements" on page 453.

Type	Compiler-Directing Statement
Library	COPY
Declarative	USE
Documentation	ENTER
Source Listing	IBM Extension *CBL *CONTROL EJECT SKIP1 SKIP2 SKIP3 TITLE End of IBM Extension

Statement Operations

COBOL statements perform the following types of operations:

- Arithmetic
- Data manipulation
- Input/output
- Procedure branching.

Common Phrases

There are several phrases common to arithmetic and data manipulation statements; these phrases are described below.

CORRESPONDING Phrase

The CORRESPONDING phrase (CORR) allows ADD, SUBTRACT, and MOVE operations to be performed on elementary data items of the same name if the group items to which they belong are specified.

Both identifiers following the key word CORRESPONDING must name group items. In this discussion, these identifiers are referred to as d1 and d2.

A pair of data items (subordinate items), one from d1 and one from d2, correspond if the following conditions are true:

- In an ADD or SUBTRACT statement, both of the data items are elementary numeric data items. Other data items are ignored.
- In a MOVE statement, at least one of the data items is an elementary item, and the move is permitted by the move rules.
- The two subordinate items have the same name and the same qualifiers up to but not including d1 and d2.
- The subordinate items are not identified by the key word FILLER.
- Neither d1 nor d2 is described as a level 66, 77, or 88 item, nor is the usage of either item INDEX or POINTER. Neither d1 nor d2 can be reference modified.
- The subordinate items do not include a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause in their descriptions; if such a subordinate item is a group, the items subordinate to it are also ignored.

  However, d1 and d2 themselves may contain or be subordinate to items containing a REDEFINES or OCCURS clause in their descriptions.

  For example, if two data hierarchies are defined as follows:

```
05  ITEM-1 OCCURS 6 INDEXED BY X.
  10  ITEM-A ...
  10  ITEM-B ...
  10  ITEM-C REDEFINES ITEM-B ...
05  ITEM-2.
  10  ITEM-A ...
  10  ITEM-B ...
  10  ITEM-C ...
```

  Then, if ADD CORR ITEM-2 TO ITEM-1(X) is specified, ITEM-A and ITEM-A(X) and ITEM-B and ITEM-B(X) are considered to be corresponding and are added together; ITEM-C and ITEM-C(X) are not included, because ITEM-C(X)

includes a REDEFINES clause in its data description. Note that d1 and d2 can be subscripted.

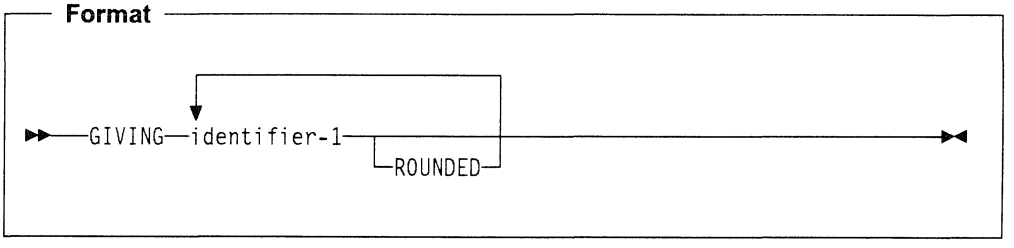
Seeing Effects of CORRESPONDING Phrase: When you use the (default) *PRTCORR compiler option or the PRTCORR option of the PROCESS statement, the compiler inserts comment lines in the compiler listing after each statement that contains the CORRESPONDING phrase. These comment lines, which print immediately before the next valid source statement, identify the elementary items that are affected within the groups named.

For examples, see the *COBOL/400 User's Guide*.

GIVING Phrase

The value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation. Because this identifier is not involved in the computation, it may be a numeric edited item.

Format



ROUNDED Phrase

After decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is compared with the number of places provided for the fraction of the resultant identifier.

When the size of the fractional result exceeds the number of places provided for its storage, truncation occurs unless ROUNDED is specified. When ROUNDED is specified, the least significant digit of the resultant identifier is increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

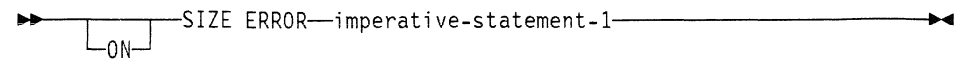
When the resultant identifier is described by a PICTURE clause containing rightmost Ps, and when the number of places in the calculated result exceeds the number of integer positions specified, rounding or truncation occurs, relative to the rightmost integer position for which storage is allocated.

SIZE ERROR Phrases

A size error condition can occur in three different ways:

- When the absolute value of the result of an arithmetic evaluation, after decimal point alignment, exceeds the largest value that can be contained in the result field
- When division by zero occurs

Format



- In an exponential expression, when

Table 24. Exponentiation Size Error Conditions

Size error	Action taken when a SIZE ERROR phrase is present
Zero raised to the exponent zero	The SIZE ERROR imperative is executed.
Zero raised to a negative exponent	The SIZE ERROR imperative is executed.
A negative number raised to a fractional exponent	The SIZE ERROR imperative is executed.

The size error condition applies only to final results, not to any intermediate results.

If the resultant identifier is defined with USAGE IS BINARY, the largest value that can be contained in it is the maximum value implied by its associated decimal PICTURE character-string.

IBM Extension

If the resultant identifier is defined with COMPUTATIONAL-4, the largest value that can be contained in it is the maximum value implied by its associated decimal PICTURE character-string.

End of IBM Extension

If the ROUNDED phrase is specified, rounding takes place before size error checking.

When a size error occurs, the subsequent action of the program depends on whether or not the ON SIZE ERROR phrase is specified.

If the ON SIZE ERROR phrase **is not** specified and a size error condition occurs, the value of the affected resultant identifier is unpredictable. Values of other resultant identifiers are not affected, as long as no size error occurred for them.

If the ON SIZE ERROR phrase **is** specified and a size error condition occurs, the value of the resultant identifier affected by the size error is not altered—that is, the error results are not placed in the receiving identifier. After completion of the execution of the arithmetic operation, the imperative statement in the ON SIZE ERROR phrase is executed, control is transferred to the end of the arithmetic statement, and the NOT ON SIZE ERROR phrase, if specified, is ignored.

For ADD CORRESPONDING and SUBTRACT CORRESPONDING statements, if an individual arithmetic operation causes a size error condition, the ON SIZE

ERROR imperative statement is not executed until all the individual additions or subtractions have been completed.

If the NOT ON SIZE ERROR phrase has been specified and, after execution of an arithmetic operation, a size error condition does not exist, the NOT ON SIZE ERROR phrase is executed.

When both ON SIZE ERROR and NOT ON SIZE ERROR phrases are specified, and the statement in the phrase that is executed does not contain any explicit transfer of control, then, if necessary, an implicit transfer of control is made after execution of the phrase to the end of the arithmetic statement.

Arithmetic Statements

The arithmetic statements are used for computations. Individual operations are specified by the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements. These operations can be combined symbolically in a formula, using the COMPUTE statement.

Arithmetic Statement Operands

The data description of operands in an arithmetic statement need not be the same. Throughout the calculation, the compiler performs any necessary data conversion and decimal point alignment.

Size of Operands: The maximum size of each operand is 18 decimal digits. The **composite of operands** is a hypothetical data item resulting from aligning the operands at the decimal point and then superimposing them on one another.

IBM Extension

The composite of operands must not contain more than 30 decimal digits.

End of IBM Extension

For example, assume that each item is defined as follows in the Data Division:

```
A  PICTURE 9(7)V9(5).
B  PICTURE 9(11)V99.
C  PICTURE 9(12)V9(3).
```

If the following statement is executed, the composite of operands consists of 17 decimal digits:

```
ADD A B TO C
```

It has the following implicit description:

```
Composite-of-Operands PICTURE 9(12)V9(5).
```

In the ADD and SUBTRACT statements, if the composite of operands is 18 digits or less, the compiler ensures that enough places are carried so that no significant digits are lost during execution. The following table shows how the composite of operands is determined for arithmetic statements:

Table 25. How the Composite of Operands is Determined

Statement	Determination of the Composite of Operands
ADD SUBTRACT	Superimposing all operands in a given statement (except those following the word GIVING)
MULTIPLY	Superimposing all receiving data items
DIVIDE	Superimposing all receiving data items, except the REMAINDER data item
COMPUTE	Restriction does not apply

In all arithmetic statements, it is important to define data with enough digits and decimal places to ensure the desired accuracy in the final result.

Overlapping Operands: When operands in an arithmetic statement share part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

Multiple Results: When an arithmetic statement has multiple results, execution conceptually proceeds as follows:

- The statement performs all arithmetic operations to find the result to be placed in the receiving items, and stores that result in a temporary location.
- A sequence of statements transfers or combines the value of this temporary result with each single receiving field. The statements are considered to be written in the same left-to-right order as the multiple results are listed.

For example, executing the following statement:

```
ADD A, B, C, TO C, D(C), E.
```

is equivalent to executing the following series of statements:

```
ADD A, B, C GIVING TEMP.
ADD TEMP TO C.
ADD TEMP TO D(C).
ADD TEMP TO E.
```

In the above example, TEMP is a compiler-supplied temporary result field. When the addition operation for D(C) is performed, the subscript C contains the new value of C.

Data Manipulation Statements

The following COBOL statements move and inspect data: ACCEPT, INITIALIZE, INSPECT, MOVE, READ, RELEASE, RETURN, REWRITE, SET, STRING, UNSTRING, and WRITE.

Overlapping Operands: When the sending and receiving fields of a data manipulation statement share a part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

Input-Output Statements

COBOL input-output statements transfer data to and from files stored on external media, and also control low-volume data that is obtained from or sent to an input/output device.

In COBOL, the unit of file data made available to the program is a record, and you need only be concerned with such records. Provision is automatically made for such operations as the movement of data into buffers and/or internal storage, validity checking, error correction (where feasible), blocking and deblocking, and volume switching procedures.

The description of the file in the Environment Division and Data Division governs which input-output statements are allowed in the Procedure Division.

All transaction file formats of the input-output statements are shown in the *COBOL/400* Reference Summary* and in the section on transaction files in the *COBOL/400 User's Guide*.

For information about COBOL file processing in relation to AS/400 file processing, see the *COBOL/400 User's Guide*.

See Appendix F, "File Structure Support Summary and Status Key Values" on page 500 for a file structure support summary.

Discussions in the following section use the terms **volume** and **reel**. The term **volume** refers to all non-unit-record input-output devices. The term **reel** applies only to tape devices. Treatment of direct access devices in the sequential access mode is logically equivalent to the treatment of tape devices.

Common Processing Facilities

There are several common processing facilities that apply to more than one input-output statement. The common processing facilities provided are:

- Status key
- INVALID KEY condition
- INTO/FROM identifier phrase
- File position indicator.

Status Key: If the FILE STATUS clause is specified in the FILE-CONTROL entry, a value is placed in the specified status key (the 2-character data item named in the FILE STATUS clause) during execution of any request on that file; the value indicates the status of that request. The value is placed in the status key before execution of any EXCEPTION/ERROR declarative or INVALID KEY/AT END phrase associated with the request.

The first character of the status key is known as status key 1 (high order digit); the second character is known as status key 2 (low order digit). The combinations of possible values and their meanings are shown in Table 59 on page 505.

For information about error handling and the role of the status key, see the chapter on exception and error handling in the *COBOL/400 User's Guide*.

INVALID KEY Condition: The invalid key condition can occur during execution of a START, READ, WRITE, REWRITE, or DELETE statement. (For details of the causes for the condition, see the appropriate statement in "Part 3. Procedure Division Statements" on page 219.) When an invalid key condition occurs, the input-output statement that caused the condition is unsuccessful.

When the invalid key condition exists after an input-output operation, actions are taken according to the type of error handling in effect:

With **standard error handling**: If there is an applicable file status clause (but not an applicable USE procedure), the file status is updated, and control returns to the program. In the absence of a file status clause, USE procedure, or INVALID KEY phrase to handle the error, a run-time message is issued, giving you the option to end or return to the program.

Without **standard error handling**: The status key, if specified, is updated. If an explicit or implicit EXCEPTION/ERROR procedure is specified for the file, the procedure runs; if no such procedure is specified, the results are unpredictable.

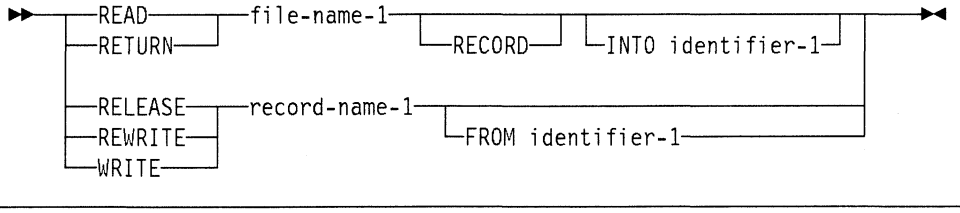
When the invalid key condition does not exist after an input-output operation, the INVALID KEY phrase is ignored, if specified, and the following actions are taken:

1. If an exception condition that is not an invalid key condition exists, control is transferred according to the rules of the USE statement following the running of any USE AFTER EXCEPTION procedure.
2. If no exception condition exists, control is transferred to the end of the input-output statement or the imperative statement specified in the NOT INVALID KEY phrase, if it is specified.

For more information about error handling and the role of the INVALID KEY phrase, see the chapter on exception and error handling in the *COBOL/400 User's Guide*.

INTO/FROM Identifier Phrase: This phrase is valid for READ, RETURN, RELEASE, REWRITE, and WRITE statements. The identifier specified must be the name of an entry in the Working-Storage Section or the Linkage Section, or of a record description for another previously opened file. Record-name/file-name and identifier must not refer to the same storage area.

Format



- The INTO phrase may be specified in a READ or RETURN statement.

  The result of the execution of a READ or RETURN statement with the INTO phrase is equivalent to the application of the following rules in the order specified:

  — The execution of the same READ or RETURN statement without the INTO phrase.

- The current record is moved from the record area to 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 in the RECORD clause. The implied MOVE statement does not occur if the execution of the READ or RETURN statement was unsuccessful. Any subscripting or reference modification associated with identifier-1 is evaluated after the record has been read or returned and immediately before it is moved to the data item. The record is available both in the record area and in identifier-1.

- The FROM phrase may be specified in a RELEASE, REWRITE, or WRITE statement.

  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:

  - The statement

    ```
    MOVE identifier-1 TO record-name-1
    ```

    according to the rules specified for the MOVE statement.

  - 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 identifier-1 is available, but the information in record-name-1 is not available, except as specified by the SAME RECORD AREA clause.

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 concept of a file position indicator has no meaning for a file opened in the output or extend mode. The setting of the file position indicator is affected only by the OPEN, READ, RETURN, ROLLBACK and START statements as follows:

- The OPEN statement positions the file position indicator to the first record in the file.

IBM Extension

The file position indicator can be positioned to any record in the file by using the POSITION parameter of the Override with database file (OVRDBF) command.

End of IBM Extension

- For a sequential access READ statement, or a dynamic access READ NEXT statement, the following considerations apply:
  - If an OPEN or START statement positioned the file position indicator, the record identified by the file position indicator is made available. If this record does not exist, the next existing record is made available.
  - If a previous READ statement positioned the file position indicator, the file position indicator is updated to point to the next existing record in the file; that record is then made available.

IBM Extension

- For a dynamic access READ FIRST statement, the file position indicator is positioned to point to the first record in the file; that record is then made available.
- For a dynamic access READ LAST statement, the file position indicator is positioned to point to the last record in the file; that record is then made available.
- For a dynamic access READ PRIOR statement, the file position indicator is positioned to point to the previous existing record in the file; that record is then made available.

End of IBM Extension

- For the RETURN statement, the following considerations apply:
  - The first RETURN statement positions the file position indicator to the first record in the file, and that record is then made available.
  - If a previous RETURN statement positioned the file position indicator, the file position indicator is updated to point to the next existing record in the file, and the record is then made available.

IBM Extension

- For the ROLLBACK statement, the following considerations apply to any file under commitment control:
  - The ROLLBACK statement sets the file position indicator to the pointer's position at the previous commitment boundary. This is important to remember if you are doing sequential processing.
  - The file position indicator is set to the pointer's position at the OPEN if no COMMIT statement has been issued since the file was opened.
  - The file position indicator is undefined for any file under commitment control that is not open.

End of IBM Extension

- The START statement positions the file position indicator to the first record in the file that satisfies the implicit or explicit comparison specified in the START statement.

The concept of the file position indicator has no meaning for files with an access mode of random or for TRANSACTION files.

IBM Extension

DB-FORMAT-NAME Special Register

After the execution of an input/output statement, for a FORMATFILE or DATABASE file, the DB-FORMAT-NAME special register is modified according to the following rules:

- After completion of a successful READ, WRITE, REWRITE, START, or DELETE operation, the record format name used in the I-O operation is implicitly moved to the special register.
- After an unsuccessful input/output operation, DB-FORMAT-NAME contains the record format name used in the last successful input/output operation.
- DB-FORMAT-NAME is implicitly defined as PICTURE X(10).

End of IBM Extension

Procedure Branching Statements

Statements, sentences, and paragraphs in the Procedure Division are executed sequentially, except when a procedure-branching statement such as EXIT, GO TO, or PERFORM is used.

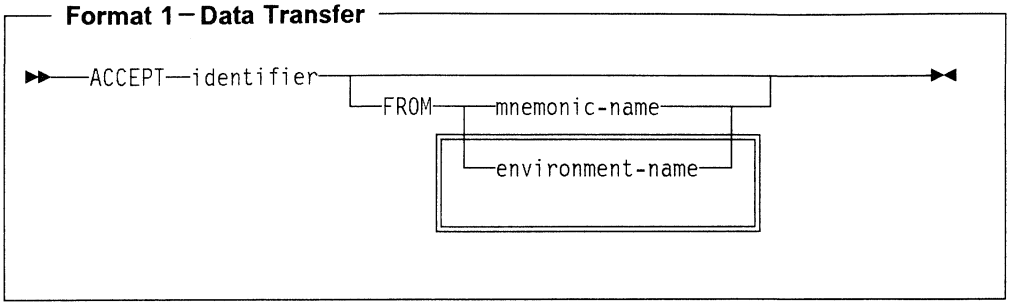
Part 3. Procedure Division Statements

ACCEPT Statement

The ACCEPT statement transfers data into the specified identifier. There is no editing or error checking of the incoming data.

Data Transfer

Format 1—Data Transfer



Format 1 transfers data from an input/output device into the identifier. No conversion is made to this data. The incoming data is received in USAGE IS DISPLAY format.

When the FROM phrase is omitted, the ACCEPT statement obtains input from the job input stream for batch jobs, and from the work station for interactive jobs.

The job input stream consists of data that accompanies a CL command. If there is no data in the input stream, an exception occurs. See "Running Your COBOL Program" in the *COBOL/400 User's Guide* for further information about the placement of input data for a batch job.

The following is an example of a batch job file member that contains a job input stream:

```
//BCHJOB   JOB(ADD021) JOBD(QUSER/ACCTEST)
CALL       PGM(QSYS/ACCPT1X)
123456789012345
//ENDBCHJOB
```

The following is an example of a COBOL program that uses a Format 1 ACCEPT statement to read the job input stream:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ACCPT1X.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-AS400.
OBJECT-COMPUTER. IBM-AS400.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  TRANS-DATA   PIC X(15).
PROCEDURE DIVISION.
BEGIN.
    ACCEPT TRANS-DATA.
    DISPLAY TRANS-DATA.
    STOP RUN.
```

When the batch job file member is used to call ACCPT1X, the ACCEPT statement reads the batch job file member from the line that immediately follows the CALL command. This causes "123456789012345" to be accepted into TRANS-DATA.

Format 1 is useful for exceptional situations in a program when operator intervention (to supply a given message, code, or exception indicator) is required. The operator must, of course, be supplied with the appropriate messages with which to reply.

identifier
The receiving data item.

FROM
mnemonic-name must be associated with an input/output device that is specified in the SPECIAL-NAMES paragraph. The input/output device can be the work station (REQUESTOR) or the system operator's message queue (CONSOLE or SYSTEM-CONSOLE). If mnemonic-name is associated with REQUESTOR and the job is a batch job, the job input stream is used.

IBM Extension

environment-name may be specified in place of mnemonic-name. Valid environment-names are CONSOLE and SYSIN.

End of IBM Extension

When the input is from the job input stream, the following rules apply:

- An input record size of 80 characters is assumed.
- If the identifier is up to 80 characters in length, the input data must appear as the first character within the input record. Any characters beyond the length of identifier are truncated.
- If the identifier is longer than 80 characters, succeeding input records are read until the storage area of the identifier is filled. If the length of the identifier is not an exact multiple of 80 characters, the last input record is truncated.

When the device is the work station, the input record size is 62. When the device is the system operator's message queue, the input record size is 58. The following steps occur:

1. A system-generated inquiry message containing the program-name, the text "AWAITING REPLY FOR POSITION(S)", and the beginning and ending positions is automatically sent to the system operator's message queue or work station operator. Previous DISPLAYs can also appear on the ACCEPT screen.
2. Processing is suspended.
3. The reply is moved into the identifier, and processing is resumed after a reply is made by the operator to the inquiry in step 1. The reply value is made available to the program as it was typed, in uppercase or lowercase.
4. If the identifier is longer than the input record size, succeeding input records are read (steps 1-3) until the identifier is filled.

If the incoming reply is longer than the identifier, the character positions beyond the length of the identifier are truncated.

Note: If the device is the same as that used for READ statements, results are unpredictable.

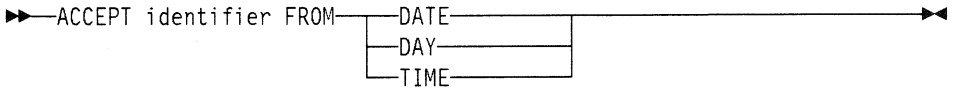
The source of input data is dependent upon the type of program initiation as follows:

Method of Program Initiation	Mnemonic-Name Associated with SYSTEM-CONSOLE	Mnemonic-Name Associated with REQUESTOR	Data Source When FROM Phrase Omitted
BATCH	System operator's message queue	Job input stream	Job input stream
INTERACTIVE	System operator's message queue	Work station	Work station

System Information Transfer

System information contained in the specified conceptual data items DATE, DAY or TIME can be transferred into the identifier. DATE, DAY and TIME are conceptual data items and, therefore, are not described in the COBOL program. The transfer must follow the rules for the MOVE statement without the CORRESPONDING phrase. See "MOVE Statement" on page 321.

Format 2—System Information Transfer



identifier
The receiving data item.

Format 2 accesses the current date (in two formats) and time of day, as carried by the system. This can be useful in identifying when a particular run of an object program was executed. It can also be used to supply the date in headings and footings.

DATE, DAY, and TIME

The conceptual data items DATE, DAY, and TIME implicitly have USAGE DISPLAY.

DATE
Has the implicit PICTURE 9(6).

The sequence of data elements (from left to right) is:

```
2 digits for year of century
2 digits for month of year
2 digits for day of month
```

Thus, 25 December 1988 is expressed as:

```
881225
```

DAY
Has the implicit PICTURE 9(5).

The sequence of data elements (from left to right) is:

```
2 digits for year of century
3 digits for day of year
```

Thus 25 December 1988 is expressed as:

```
88360
```

TIME

Has the implicit PICTURE 9(8).

The sequence of data elements (from left to right) is:

```
2 digits for hour of day
2 digits for minute of hour
2 digits for second of minute
2 digits for hundredths of second
```

Thus 12.25 seconds after 2:41 PM is expressed as:

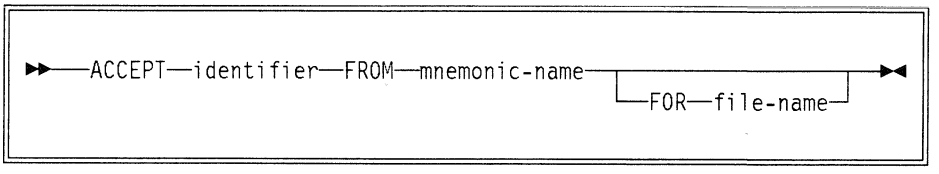
```
14411225
```

IBM Extension

Feedback

This format is used to transfer feedback information from an active file to the identifier.

Format 3—Feedback



The identifier can be any fixed-length group item or an elementary alphabetic, alphanumeric, or external decimal item. The file must be defined in an FD entry, and must be open prior to the execution of the ACCEPT statement. If the file is not open, the contents of identifier remain unchanged.

The FROM phrase specifies a mnemonic-name that must be associated with an environment-name of OPEN-FEEDBACK or I-O-FEEDBACK in the SPECIAL-NAMES paragraph.

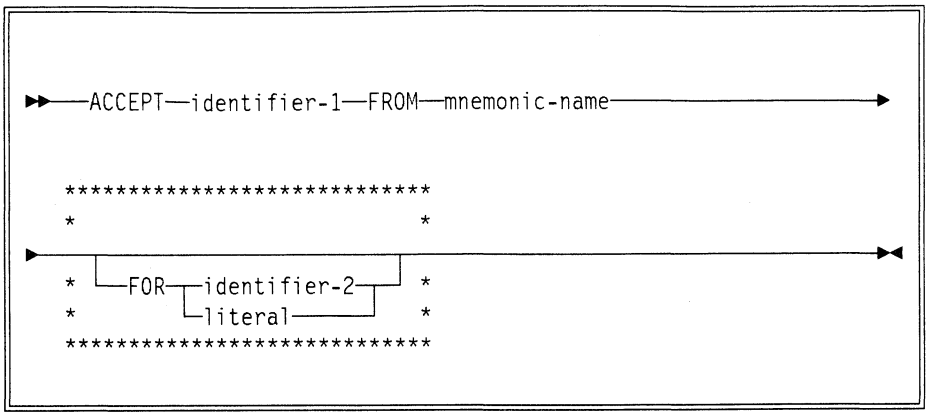
When the FOR phrase is specified, the feedback information is from the file specified in the phrase. When the FOR phrase is not specified, the feedback information is from the last file opened or used in an input or output operation.

See Appendix F, "File Structure Support Summary and Status Key Values" on page 500 for a discussion of the I-O-FEEDBACK and OPEN-FEEDBACK areas. See the *Data Management Guide* for a layout and description of the data areas contained in the feedback areas.

Local Data Area

This format is used to transfer data to identifier-1 from the system-defined local data area created for a job.

Format 4—Local Data Area



This format is only applicable when the mnemonic-name in the SPECIAL-NAMES paragraph is associated with the environment-name LOCAL-DATA.

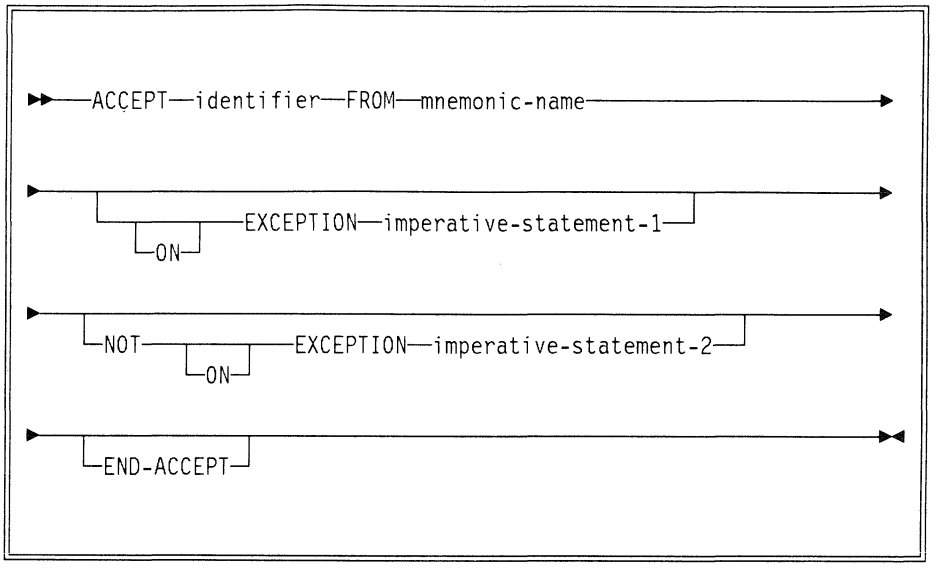
The move into identifier-1 takes place according to the rules for the MOVE statement for a group move without the CORRESPONDING phrase.

When the FOR phrase is specified, it is syntax checked during compilation but treated as a comment during execution. The value of literal or identifier-2 indicates the program device name of the device that is associated with the local data area. There is only one local data area for each job, and all devices in a job access the same local data area. Literal, if specified, must be nonnumeric and 10 characters or less in length. Identifier-2, if specified, must refer to an alphanumeric data item, 10 characters or less in length. For more information about the local data area, see the *CL Programmer's Guide* and the *COBOL/400 User's Guide*.

Format 5 Considerations

You use this format to transfer data from the PIP (Program Initialization Parameters) data area into the identifier.

Format 5—PIP Data Area



This format only applies when you associate the mnemonic-name in the SPECIAL-NAMES paragraph with the environment-name PIP-DATA.

The move into the identifier takes place according to the rules for the MOVE statement for a group move without the CORRESPONDING phrase.

If the PIP data area exists, the job is a prestart job, and any imperative statement specified in the NOT ON EXCEPTION phrase is processed. If the PIP data area does not exist, the job is not a prestart job, and any imperative statement specified in the ON EXCEPTION phrase is processed. In the absence of the ON EXCEPTION phrase, a run-time message is issued if the PIP data area does not exist.

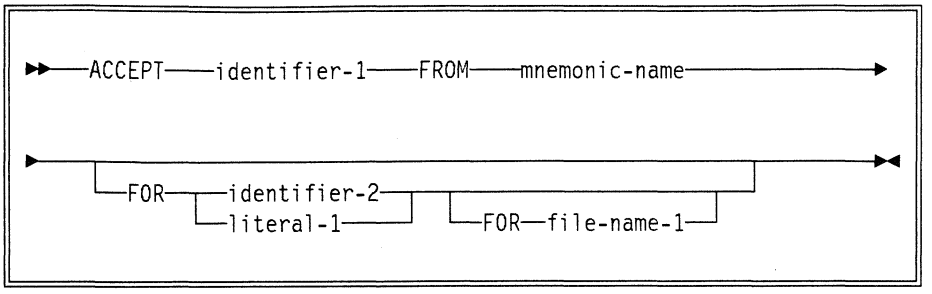
The END-ACCEPT explicit scope terminator serves to delimit the scope of the ACCEPT statement. END-ACCEPT permits a conditional ACCEPT statement to be nested in another conditional statement. END-ACCEPT may also be used with an imperative ACCEPT statement. For more information, see "Delimited Scope Statements" on page 207.

Note that you cannot update the PIP data area using COBOL. For more information about the PIP data area, see the *COBOL/400 User's Guide* and the *CL Programmer's Guide*.

Attribute Data

The ACCEPT statement retrieves information (attribute data) about a particular program device associated with a TRANSACTION file.

Format 6—Attribute Data



This format of the ACCEPT statement may only be used for files with an organization of TRANSACTION. If the file is not open at the time the ACCEPT is executed, message LBE7205 is issued and execution terminates. **Mnemonic-name** must be associated with the environment-name ATTRIBUTE-DATA in the SPECIAL-NAMES paragraph.

If **file-name** is not specified, the default file for the ACCEPT statement is the first TRANSACTION file specified in a SELECT clause of the FILE-CONTROL paragraph.

Literal-1 or the contents of **identifier-2**, if specified, indicates the program device name for which attribute data is made available.

For an ICF file, this device must have been defined (through a ADDICFDEVE, CHGICFDEVE, or OVRICFDEVE command) as available to be acquired by the file, but need not have actually been acquired. For a display file, if the program device name is not the name of the display device, then the device must have been specified in the DEV parameter when the file was created, changed, or overridden, and before the OPEN is issued for the file. Literal, if specified, must be nonnumeric and 10 characters or less in length. The contents of identifier-2, if specified, must be an alphanumeric data item 10 characters or less in length. If an invalid program device name is specified, message LBE7205 is issued and execution terminates.

If both FOR phrases are omitted (indicating the default TRANSACTION file is being used) the ACCEPT statement uses the program device from which a READ, WRITE, REWRITE, or ACCEPT (Attribute Data) operation on the default file was most recently performed. If the only prior operation on the file was an OPEN, the ACCEPT statement uses the program device implicitly acquired by the file when the file was opened. When both FOR phrases are omitted, a program device must have been acquired in order to use this format of the ACCEPT statement. See the *ICF Programmer's Guide* and the *Data Management Guide* for more information on acquiring devices.

Program device attributes are moved into identifier-1 from the appropriate attribute data format, according to the rules for a group MOVE without the CORRESPONDING phrase.

Attribute Data Formats

The attribute data retrieved by the ACCEPT statement depends on whether the data and the associated fields are applicable to a work station or to a communications device. See Appendix F, "File Structure Support Summary and Status Key Values" on page 500 for format descriptions.

The ATTRIBUTE-DATA mnemonic name can be used *only* to obtain information about a program device acquired by a TRANSACTION file. Attribute data does *not* provide information about the status of a completed or attempted I-O operation. To obtain information about I-O operations, use the Format 3 ACCEPT statement with the I-O-FEEDBACK or OPEN-FEEDBACK mnemonic names.

Workstation I/O

You can use this format of the ACCEPT statement **only** if you specify the *EXTACCDSP generation option of the CRTCBLPGM command, or the EXTACCDSP option of the PROCESS statement. See the *COBOL/400 User's Guide* for information about these options.

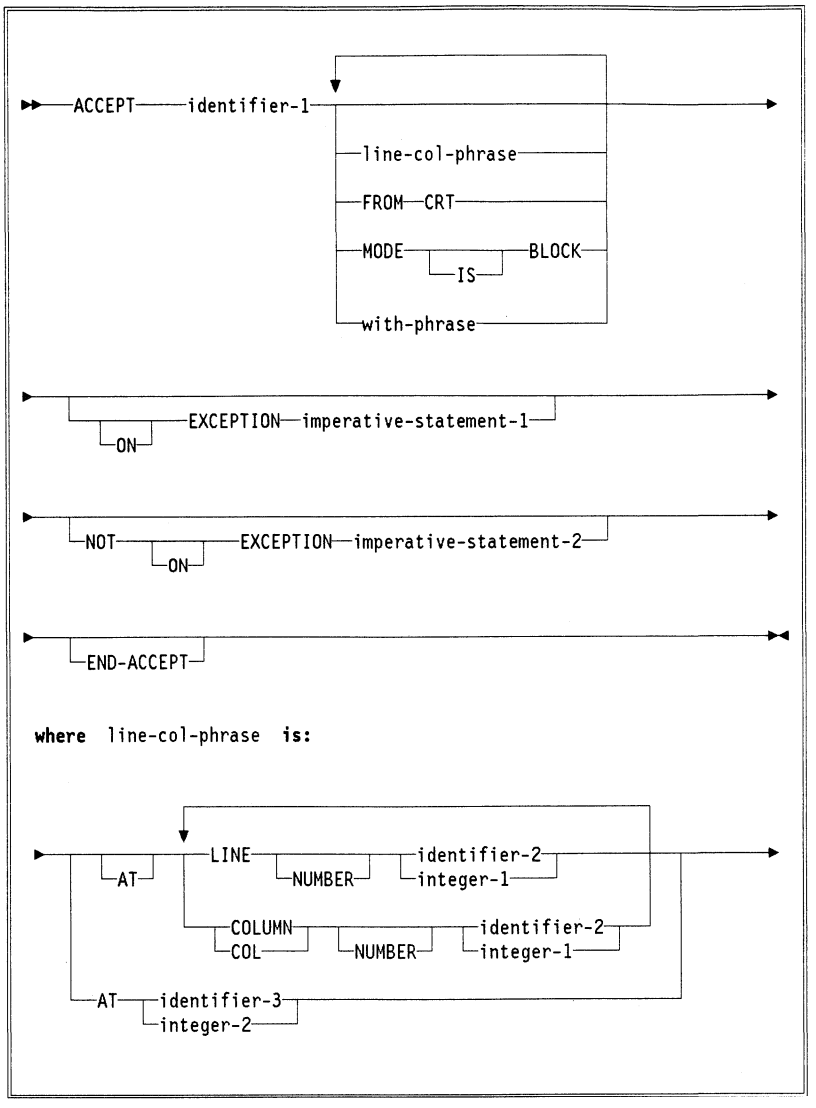
An ACCEPT statement is considered an **extended** ACCEPT statement if it:

- has an AT phrase, or
- has a FROM phrase with the CRT option, or
- has a MODE IS BLOCK phrase, or
- has a WITH phrase, or
- has an ON EXCEPTION phrase or a NOT ON EXCEPTION phrase, (and PIP-DATA is not specified for mnemonic-name), or
- does not have a FROM phrase, but CONSOLE IS CRT is specified in the SPECIAL-NAMES paragraph.

An ACCEPT statement is considered a **standard** ACCEPT statement if it:

- has a FROM phrase (other than FROM CRT) and CONSOLE IS CRT is specified in the SPECIAL-NAMES paragraph, or
- does not have a FROM phrase and CONSOLE IS CRT is **not** specified.

Format 7—Workstation I/O

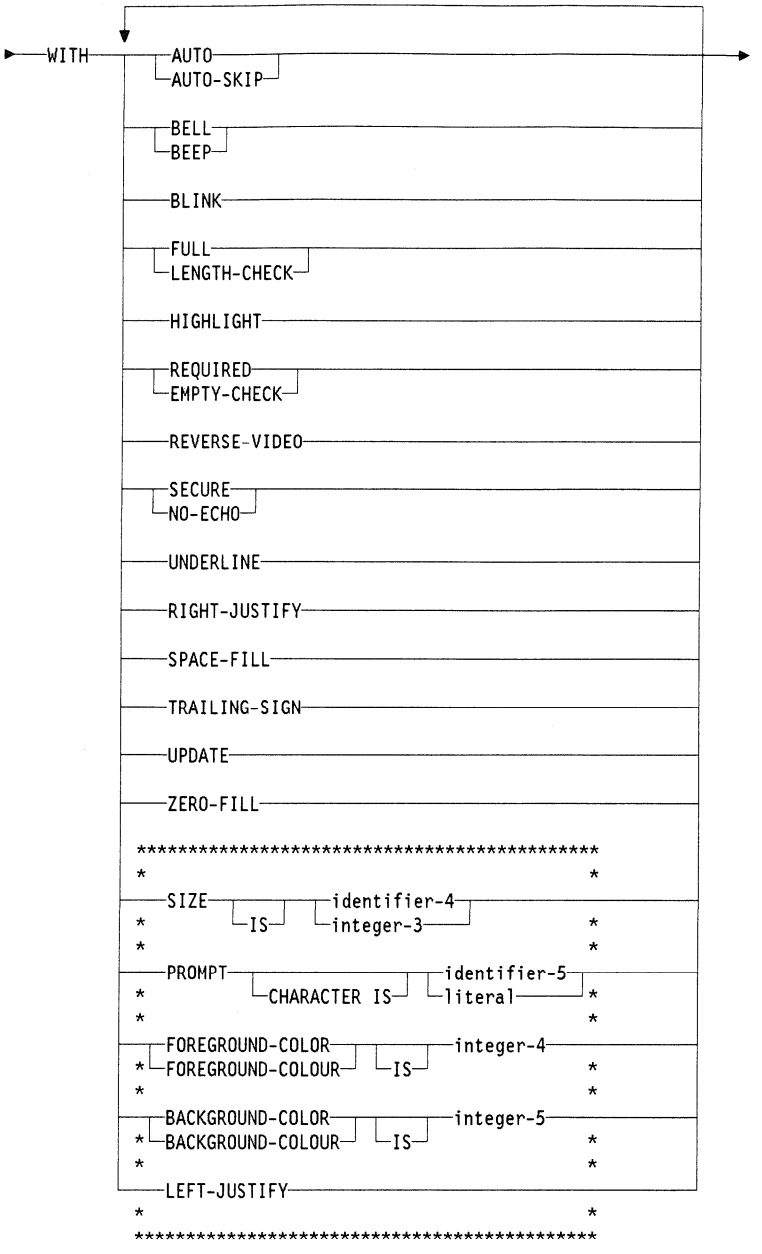


(continued on next page)

Format 7—Workstation I/O

(continued from previous page)

where with-phrase **is:**

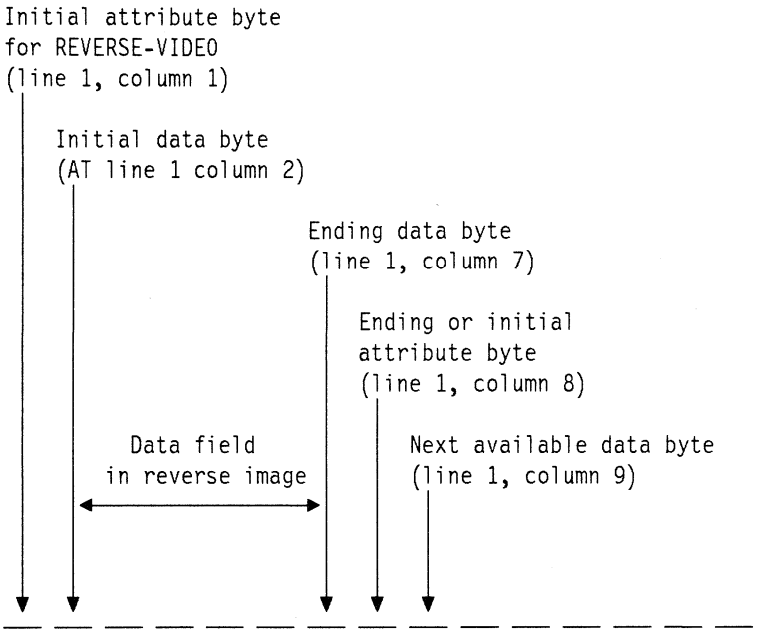


identifier-1

A data item whose value may be updated.

Fields accepted or displayed require an attribute byte before and after the field. To accomplish this, space must be available on the screen for, at a minimum, the initial display attribute. For this reason, line 1 and column 1 cannot be used for data because that position is required for the first display attribute. The lowest position that can be used on the screen for data is line 1, column 2.

For example:



The AT phrase sets the starting line and column for the fields that will be accepted or displayed. It does not indicate the position of the initial display attribute.

It is the user's responsibility to ensure that each field is positioned on the screen to prevent attribute bytes from overlaying data bytes, and to prevent data bytes from overlaying attribute bytes. The user should also be aware that the ending attribute byte will be the normal attribute defined for the specific work station. Therefore, the user should ensure that the attributes are specified in the correct order to obtain the expected results.

The user should initially clear the screen by using a DISPLAY statement that contains the WITH BLANK SCREEN phrase.

If identifier-1 is a group item and there is no MODE IS BLOCK phrase, those elementary subordinate items that have names other than FILLER are displayed. They are displayed simultaneously and positioned on the screen in the order that their descriptions appear in the DATA DIVISION, separated by the lengths of the FILLER items in the group. For this purpose, the first position on a line is regarded as immediately following the last position on the previous line.

When items are separated by FILLERs, the attribute bytes are included in the FILLER length, so a FILLER of one or two bytes would contain both the trailing and leading attributes of separate items. In the case of a one-byte

FILLER, the trailing and leading attributes would occupy the same byte. Since data items are normally separated by one attribute byte, one-byte FILLER items are not necessary.

The extended ACCEPT statement predisplays the value of identifier-1, accepting the value from the workstation after the ENTER key is pressed.

The accepted value is the image of the characters taken from the screen positions designated to the data item, and delimited by the cursor position at the time the FIELD EXIT key is pressed. Simply pressing the ENTER key causes all of the predisplayed characters to be accepted.

The FIELD EXIT key will convert all trailing spaces in the alphabetic and alphanumeric fields to hexadecimal zeroes. For example:

```
DATA DIVISION.
       01    STRUC1.
             03  F11 PIC AA VALUE 'A'.
             03  F12 PIC 9(4) VALUE 123.
             03  F13 PIC XXX VALUE 'B'.

PROCEDURE DIVISION.
       ACCEPT STRUC1 AT 2102.
```

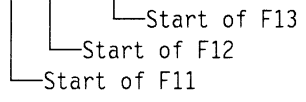
The ACCEPT statement in the example predisplays three fields:

- F11 starts at line 21, column 2. Its attribute byte occupies line 21, column 1.
- F12 starts at line 21, column 5. Its attribute byte occupies line 21, column 4.
- F13 starts at line 21, column 10. Its attribute byte occupies line 21, column 9.

Here is the visual result at line 21:

```
00000000011111111112
12345678901234567890

A   123 B
```



Note that numeric field F12 is right justified, and its first position consists of a space.

When you place the cursor at the end of field F13 and press the FIELD EXIT key in response to the values of the three fields, the values of the fields remain unchanged. Trailing spaces (X'40') in F11 and F13 do, however, become hexadecimal zeroes, so the test IF F13='Bbb' will fail after the ACCEPT operation.

To avoid trailing hexadecimal zeroes, use the SPACE-FILL phrase. If existing programs already include extended ACCEPT statements without this phrase, and program changes are costly or not the desired solution, you can consider using the *NOUNDSPCHR compiler option. Use this option exclusively for programs in which the extended ACCEPT and extended DISPLAY statements handle displayable characters only.

The displayable-only characters will be handled if:

- Extended DISPLAY statements do not use binary or packed data (directly or redefined).
- Extended ACCEPT statements do not use binary or packed data redefined as alphanumeric fields.

Note: The MODE IS BLOCK phrase is an implied redefinition of a data group to an alphanumeric field.

For example:

```
01     STRUC2.
       03  F21 PIC 99.
       03  F22 PIC 9(10) USAGE COMP-3 VALUE 1111123.
       03  F23 PIC X(5).
```

ACCEPT STRUC2 MODE IS BLOCK AT 0102 will contain undisplayable characters because it will be handled as one alphanumeric field 13 bytes long.

Data Categories: The following table shows the data categories handled by the extended ACCEPT statement. These data categories are also supported by the extended DISPLAY statement.

Table 26. Data Categories Handled by Extended ACCEPT

Category	Initial Display	Entering Data	Data Item Updated
Alphabetic	A, B, C	D	B, C
Numeric (binary)	B, C, E, F, F1, F2	D, G, H	O
Numeric (internal decimal or packed decimal)	B, C, E, F, F1, F2	D, G, H	O
Numeric (zoned decimal)	B, C, F, F1, F2	D, G, H	O
Numeric-edited	A, I	J, H	K, O
Alphanumeric	A, B, C	D	B, C
Alphanumeric-edited	A, I	J	L
Boolean	A, B, C	D, M, N	B, C

A Left justified (by default).

B For RIGHT-JUSTIFY, trailing spaces and hexadecimal zeros are removed, and data is moved to the rightmost position.

C For ZERO-FILL, trailing spaces and hexadecimal zeros are converted to zeros if data is left-justified. Leading spaces are converted to zeros if data is right-justified.

If only SPACE-FILL is specified, trailing hexadecimal zeros are converted to spaces.

D If RIGHT-JUSTIFY and ZERO-FILL (or SPACE-FILL) are specified, the work station will right-justify and zero-fill or space-fill the field when the field exit key is pressed.

If only ZERO-FILL (or SPACE-FILL) is specified, the work station does not make any conversions.

E A binary or packed number is converted to zoned decimal before it is displayed.

F The following conditions occur:

- The number is padded with spaces on the left before it is displayed.
- Initially, the decimal point is inserted for decimal digits, to divide the integer from the fractional digits.
- One position is reserved for the decimal point if there are fractional digits.
- If DECIMAL-POINT IS COMMA is specified in the SPECIAL-NAMES paragraph, then the meanings of comma and period are reversed. Note, there is no association with the system value QDECFMT.
- One position is reserved for the sign, if a number is signed.
- If the number has a negative sign, the sign is displayed. By default, the sign is leading.

F1 When ZERO-FILL is specified, leading zeros are displayed.

F2 When TRAILING-SIGN is specified, the sign occupies the rightmost position.

G Digits, blanks, and the following symbols are accepted:

– (minus)
+ (plus)
. (period)
, (comma)

The sign must be entered in the leading or trailing position.
The decimal point must be entered before the fractional digits.
Digits are not justified. A comma separates each group of three integer digits.

H A number has the following characteristics:

1. The sign symbol value is optional and if present, may precede any digit value (a leading sign) or may follow the digit value (a trailing sign). Valid signs are positive and negative. The sign symbol, if it is a leading sign, may be preceded by blank characters. If the sign symbol is a trailing sign, it must be the rightmost character in the field. Only one sign symbol is allowed.
2. Up to 31 decimal digits may be specified. Valid decimal digits are in the range 0 through 9. The first decimal digit may be preceded by blank characters but blank characters located to the right of the leftmost decimal digit are not valid.

   The decimal digits may be divided into two parts: an integer part and a fractional part. Digits to the left of the decimal point are interpreted as integer values. Digits to the right are interpreted as fractional values. If no decimal point symbol is included, the value is interpreted as an integer value. If the decimal point symbol precedes the leftmost decimal digit, the digit value is interpreted as a fractional value, and the leftmost decimal digit must be adjacent to the decimal point symbol. If the decimal point follows the rightmost decimal digit, the digit value is interpreted as an integer value, and the rightmost decimal digit must be adjacent to the decimal point.

   Decimal digits in the integer portion may optionally have comma symbols separating groups of three digits. The leftmost group may contain one, two, or three decimal digits, and each succeeding group must be preceded by the comma symbol and contain three digits. The comma symbol must be adjacent to a decimal digit on either side.

   Decimal digits in the fractional portion may not be separated by commas and must be adjacent to one another.

I The number is edited according to the PIC symbol specified. ZERO-FILL, SPACE-FILL, and RIGHT-JUSTIFY do not affect an edited field.

J Data should be entered with edited symbols.

K All editing symbols are removed, then the resulting number is moved back with editing into the numeric-edited field. A run-time message will be issued if nonnumeric characters are detected.

L Data is moved back into the field and **no** editing is performed. It is the user's responsibility to ensure that the edited format is followed.

M Digits, blanks, and the following symbols are accepted:

- (minus)
+ (plus)
. (period)

, (comma)

N Any character that is not a zero or a one will generate an error message.

O The numeric field is aligned on the assumed decimal position. See "Alignment Rules" on page 107 for the rules about positioning data.

The phrases following identifier-1 can be in any order. All phrases specified apply to the previous identifier.

AT Phrase

The AT phrase indicates the absolute address on the screen at which the ACCEPT operation is to start. If the AT phrase is not specified, the ACCEPT operation starts at line 1, column 2. It does not indicate the starting position of the leading attribute.

The LINE phrase specifies the line at which the screen item starts on the screen.

The COLUMN phrase specifies the column at which the screen item starts on the screen.

The LINE and COLUMN phrases can appear in any order.

COL is an abbreviation for COLUMN.

identifier-2
integer-1

Identifier-2 and integer-1 must be an unsigned numeric integer with a value greater than or equal to zero. If the value for LINE or COLUMN is negative, the absolute value is taken. Identifier-2 or integer-1 is moved into a PIC 9(3) number.

Certain combinations of line and column numbers have special meaning:

- Until the column comes within range, out of range column values are reduced by the line length and the line value is incremented. A column number may cause the line number to be incremented several times.
- Out of range line values cause the screen to scroll up one line. The effect is the same as if the line number of the bottom line had been specified. The screen is never scrolled more than one line up regardless of the line specified.
- If column and line numbers are both out of range, out of range columns are handled first followed by out of range lines (according to rules above).
- If the line and column numbers given are both zero, the ACCEPT starts at the position following that where the preceding ACCEPT operation finished. Column 1 of each line is considered to follow the last column of the previous line.
- If the line number is zero, but a non-zero column number is specified, the ACCEPT starts at the specified column, on the line following that where the preceding display operation finished.
- If the column number is zero, but a non-zero line number is specified, the ACCEPT starts on the specified line, at the column following that where the preceding display operation finished.

identifier-3
integer-2

Identifier-3 must be a PIC 9(4) or a PiC 9(6) field. Integer-2 must be a 4- or 6-byte numeric field.

If identifier-3 or integer-2 is 4 digits long, the first 2 digits specify the line, and the second 2 digits specify the column. If identifier-3 or integer-2 is 6 digits long, the first 3 digits specify the line, the second 3 digits specify the column.

FROM CRT Phrase

Indicates that the ACCEPT statement is extended.

MODE IS BLOCK Phrase

The identifier is to be treated as an elementary item; thus, even if it is a group item it is accepted as one item.

ON EXCEPTION Phrases

If ON EXCEPTION is specified, imperative-statement-1 is executed if the ACCEPT operation finishes with anything other than a normal completion. That is, if the CRT Status Key 1 is other than 0.

The use of the ON EXCEPTION phrase does not prevent the generation of a run-time message for such conditions as work station boundaries or out-of-screen ranges.

If NOT ON EXCEPTION is specified, imperative-statement-2 is executed if the ACCEPT operation finishes with a normal completion.

END-ACCEPT Phrase

END-ACCEPT is optional. It is required if ACCEPT statements are nested.

WITH Phrase

The WITH phrase allows the user to specify certain options for the ACCEPT operation. These options are described in the following phrases.

AUTO (AUTO-SKIP) Phrase

When a field has been filled by operator input, the cursor automatically steps to the next input field, rather than waiting for a terminating character to be entered. If the field is the last in a group, AUTO-SKIP acts as if the ENTER key had been pressed.

AUTO and AUTO-SKIP may be used interchangeably.

BELL (BEEP) Phrase

An audible alarm sounds each time the item containing this phrase is accepted.

BELL and BEEP may be used interchangeably.

BLINK Phrase
The screen item blinks when it appears on the screen.

FULL (LENGTH-CHECK) Phrase
The operator must either leave the screen item completely empty or fill it entirely with data. The FIELD-EXIT, FIELD+, FIELD- keys are not allowed. Any attempt to use the delete key on the data within the input field, followed by the enter key, is also not allowed. The FULL phrase can be satisfied by data that is initially displayed.

If this phrase is specified at a group level, it applies to all suitable subordinate elementary items.

The FULL phrase is effective during the execution of any ACCEPT statement.

FULL and LENGTH-CHECK may be used interchangeably.

HIGHLIGHT Phrase
The screen item is in high-intensity mode when it appears on the screen.

REQUIRED (EMPTY-CHECK) Phrase
The REQUIRED phrase is used to ensure that a field does not remain empty.

For alphanumeric items, this means that the field must contain at least one character other than a space or a hexadecimal zero. For numeric items, the field must contain a value of other than zero.

If a field remains empty when this phrase is specified, a run-time message will be issued which requires the user to press the reset key and then to re-enter the data.

REQUIRED and EMPTY-CHECK may be used interchangeably.

REVERSE-VIDEO Phrase
The screen item is displayed in reverse image.

SECURE (NO-ECHO) Phrase
Operator-keyed data is prevented from appearing on the screen. This phrase may be specified on a group screen item, in which case it applies to all suitable elementary items which are subordinate to that item. When the SECURE phrase is specified, only spaces and cursor appear in the screen item.

SECURE and NO-ECHO may be used interchangeably.

UNDERLINE Phrase
The screen item is underlined when it appears on the screen.

RIGHT-JUSTIFY Phrase
Operator-keyed characters are moved on the screen to the rightmost character positions of the field. Trailing spaces and trailing hexadecimal zeros are removed.

This option affects only non-edited data items. This takes effect upon display of the initial data in the data item and also upon termination of the ACCEPT operation. This is the only way in which numeric data are handled.

If the data item is defined with the JUSTIFIED RIGHT clause in the DATA DIVISION, then the data item is treated as if the RIGHT-JUSTIFY phrase had been specified.

SPACE-FILL Phrase

For non-edited data items, trailing hexadecimal zeros are converted to spaces, and the items appear on the screen with zero-suppression in all character positions. This takes effect when initial data in the data item is displayed and again when the ACCEPT operation into the data item is terminated. This option has no effect on edited fields.

TRAILING-SIGN Phrase

The operational sign appears in the rightmost character position of the field. This takes effect upon display of initial data in the data item and also upon termination of the ACCEPT operation. This option affects only signed, non-edited numeric data items. When this option is not specified, the sign precedes the number.

UPDATE Phrase

The current contents of the data item are displayed before the operator is prompted to key in any new data; the initial data is then treated as though it were operator-keyed.

Predisplaying by Data Type: In the absence of the UPDATE phrase, you can control the predisplaying of some data. To predisplay only numeric-edited data, specify the *ACCUPDNE option of the EXTDSPOPT parameter. To predisplay all data, use the default option, *ACCUPDALL.

ZERO-FILL Phrase

Non-edited data items appear on the screen with no zero-suppression. For left-justified data, trailing spaces and trailing hexadecimal zeros are converted to zeros. For right-justified data, leading spaces are converted to zeros.

This takes effect when initial data in the data item is displayed and again when the ACCEPT operation into the data item is terminated. It has no effect on edited fields.

Phrases Syntax Checked Only

The following phrases are syntax checked but are treated as documentation by the compiler.

SIZE
PROMPT CHARACTER
FOREGROUND-COLOR or FOREGROUND-COLOUR
BACKGROUND-COLOR or BACKGROUND-COLOUR
LEFT-JUSTIFY

Format 7 Considerations

If identifier-1 is a group item and there is no MODE IS BLOCK phrase, those elementary subordinate items that have names other than FILLER are accepted. They are positioned on the screen in the order that their descriptions appear in the DATA DIVISION, and are separated by the lengths of the FILLER items in the group.

For this purpose, the first position on a line is regarded as immediately following the last position on the previous line. The items are accepted in the same order.

Unless otherwise specified in the CURSOR clause, the cursor initially points at the start of the first item. As the ACCEPT operation into each item terminates, the cursor moves to the start of the next item.

The CURSOR clause has no effect on the position of the fields; it can only change the cursor position for the ACCEPT statement according to stated rules.

Numeric items with PICTURE clauses containing the symbol P are not supported by the extended ACCEPT statement.

Unless you specify MODE IS BLOCK, data items must not contain fixed-length tables. Data items must not contain variable-length tables whether or not you specify MODE IS BLOCK.

The LEFT-JUSTIFY phrase is syntax checked only. It has no effect on how data is positioned. Numeric data is always right justified. LEFT-JUSTIFY is accepted but has no effect on the way in which numeric data is handled.

Extended ACCEPT and Extended DISPLAY Considerations

The following considerations are common to both the extended ACCEPT and the extended DISPLAY statements.

Number of Files

Since a display file, QDLBACCDSP, is used by COBOL for the extended ACCEPT and DISPLAY statements, the user can specify only 98 files in a program (rather than the 99 that were allowed in releases prior to Version 1, Release 3).

If extended ACCEPT and DISPLAY statements are not used, the user may code 99 files.

Storage

The total size for a single user data stream (more than one may be created by a single extended ACCEPT or DISPLAY statement) is 3 000 bytes. An error (LBL1294) occurs if this limit is reached.

The limit is due to the size (number of bytes of storage) of all data items ACCEPTed or DISPLAYed in a statement, plus, all COBOL/400 internal structures necessary to position the items on the screen and interpret all of the phrases specified.

Depending on the phrases specified, the COBOL/400 compiler may use a significant amount of storage for internal structures, leaving less storage to resolve data items during an ACCEPT or DISPLAY operation.

Screen Format

Extended ACCEPT and DISPLAY operations support a 24-line by 80-column screen format.

When extended ACCEPT or DISPLAY operations are processed, no other display file should be open by the program. If TRANSACTION files are coded in a program that contains extended ACCEPT or DISPLAY statements, it is the user's responsibility to ensure that TRANSACTION I/O does not interfere with extended ACCEPT or DISPLAY statements. Conversely, the user should ensure that extended ACCEPT or DISPLAY statements do not interfere with TRANSACTION I/O operations.

Reserved Words

Extended ACCEPT and DISPLAY operations define reserved words that might already be user-defined. The *EXTACCDSP generation option of the CRTCBLPGM command can help you identify these words.

When *EXTACCDSP is specified, words that might be user-defined or flagged as not used by the COBOL/400 compiler, but which are used by other SAA compilers, become reserved words.

If the default, *NOEXTACCDSP, is specified, the reserved words normally defined by the COBOL/400 compiler are used. See Appendix E, "COBOL/400 Reserved Word List" on page 497 for an explanation and a list of the reserved words.

For syntax-checking purposes, the COBOL/400 syntax checker assumes that *EXTACCDSP has been specified.

Release Availability

Extended ACCEPT and DISPLAY statements cannot be compiled using the TGTRLS(*PRV) parameter in Version 1, Release 3, or using any release of the COBOL/400 compiler prior to Version 1, Release 3. See the *COBOL/400 User's Guide* for a description of Target Release (TGTRLS).

Subscripting and Reference Modification

Subscripted items are supported, but reference modified items are not.

Performance

Unless you specify the *NODFRWRT (no deferred writing) option of the CRTCBLPGM EXTDSPOPT parameter, the COBOL/400 compiler buffers all extended DISPLAY statements until the next ACCEPT statement is encountered. While the *NODFRWRT option allows you to associate data errors with the statements that cause them by performing DISPLAY statements as they are encountered, the deferred writing (*DFRWRT) option improves performance by buffering data streams generated by consecutive DISPLAY statements.

DBCS Processing

DBCS programs can run on a DBCS system only if they have been compiled on a DBCS system:

- The user must code shift-in and shift-out characters properly to permit the continuation of DBCS items. See the appendix on Double-Byte Character Set support in the *COBOL/400 User's Guide* for the rules about continuing DBCS items.
- DBCS content is governed by the rules discussed in the appendix on Double-Byte Character Set support in the *COBOL/400 User's Guide*.
- For alphabetic, alphanumeric, and alphanumeric-edited fields, both regular and extended DBCS characters are supported; however, results will not be predictable for single-byte characters in those fields whose value is less than hexadecimal 40.
- For a group item with MODE IS BLOCK, the user can display any type of data in the structure. That is, hexadecimal values less than 40 are accepted. Regular DBCS characters are supported. When extended DBCS characters are used, a symbol is displayed to indicate that the characters are not defined.
- When the length of an alphabetic or an alphanumeric field is less than 4 bytes, an error is not generated if a value of less than hexadecimal 40 is encountered.

Combinations of Phrases

When one ACCEPT or DISPLAY statement contains the UNDERLINE, HIGHLIGHT and REVERSE-VIDEO phrases in one WITH phrase, the HIGHLIGHT phrase is ignored. A warning message (LBL0265) is generated at compilation time if this combination is coded. In an extended DISPLAY statement, the UPON CRT-UNDER phrase is equivalent to the UNDERLINE phrase. To protect a field from being displayed on the screen, use the SECURE option.

Display File QDLBACCDSP

This file is created by the compiler for use by extended ACCEPT and DISPLAY statements. This file is created with the DDS keyword KEEP, and with these Create Display File (CRTDSPF) parameter values: a value of *YES for Restore Display (RSTDSP), and a value of *NO for Defer Write (DFRWRT). See the *CL Reference* for an explanation of the Create Display File command and its parameters.

The CANCEL statement does not close this file.

Remote Workstations

Extended ACCEPT and extended DISPLAY statements do not run on remote workstations attached to 5251 Model 12 controllers.

The *NOUNDSPCHR option of the CRTCBLPGM EXTDSPOPT parameter allows you to use extended ACCEPT and extended DISPLAY statements at remote workstations attached to 3174 and 3274 controllers, provided that your data does not contain undisplayable characters.

For more information about this option, see the *COBOL/400 User's Guide*.

Differences from COBOL/2 Processing

The COBOL/400 extended ACCEPT and DISPLAY statements are similar to the IBM COBOL/2* ACCEPT and DISPLAY statements (Format 2). The exceptions are discussed in Appendix I, "ACCEPT/DISPLAY and COBOL/2 Considerations" on page 522.

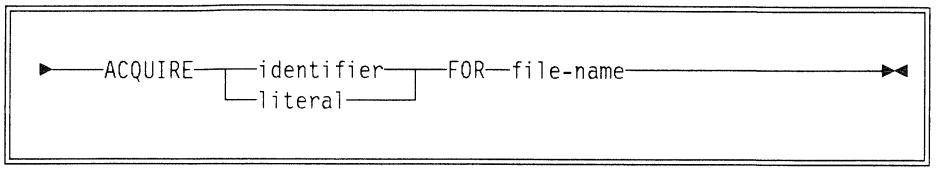
End of IBM Extension

IBM Extension

ACQUIRE Statement

The ACQUIRE statement acquires a program device for a TRANSACTION file.

Format—TRANSACTION



identifier
literal

The literal you specify, or the contents of the identifier, will specify the program device name to be acquired by the specified file. Literal, if specified, must be nonnumeric and 10 characters or less in length. Identifier, if specified, must refer to an alphanumeric data item 10 characters or less in length.

file-name

File-name must be the name of a file with an organization of TRANSACTION, and the file must be open when the ACQUIRE statement is run. A compilation error message is issued if the organization is not TRANSACTION.

For a description of conditions that must be met before a device can be acquired, see the *Data Management Guide* for display stations, and the *ICF Programmer's Guide* for communication devices.

Successful completion of the ACQUIRE operation makes the program device available for input and output operations. If the ACQUIRE is unsuccessful, the file status value is set to 9H and any applicable USE AFTER EXCEPTION/ERROR procedure is invoked.

Only one program device may be implicitly acquired when a file is opened. If a file is an ICF file, the single implicitly acquired program device is determined by the ACQPGMDEV parameter of the CRTICFF CL command. If the file is a display file, the single implicitly acquired program device is determined by the first entry in the DEV parameter of the CRTDSPF CL command. Additional program devices *must* be explicitly acquired.

A program device is explicitly acquired by using the ACQUIRE statement. For an ICF file, the program device must have been defined to the file with the ADDICFDEVE or OVRICFDEVE command before the file is opened. For a display file, if the program device name is not the name of the display device, then the device must have been specified in the DEV parameter when the file was created, changed, or overridden, and before the OPEN is issued for the file.

For more information on these CL commands, see the *CL Reference* and the *Data Management Guide* for display stations, and the *ICF Programmer's Guide* for communication devices.

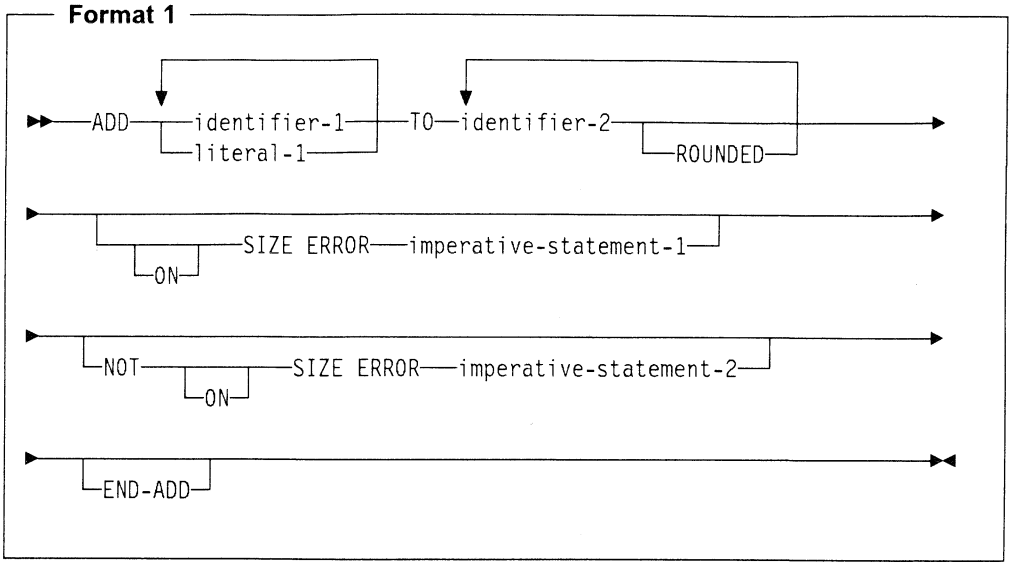
The ACQUIRE statement can also be used as an aid in recovering from I-O errors. For more information on recovery procedures, see the section on "Communications Recovery" in the *COBOL/400 User's Guide*.

End of IBM Extension

ADD Statement

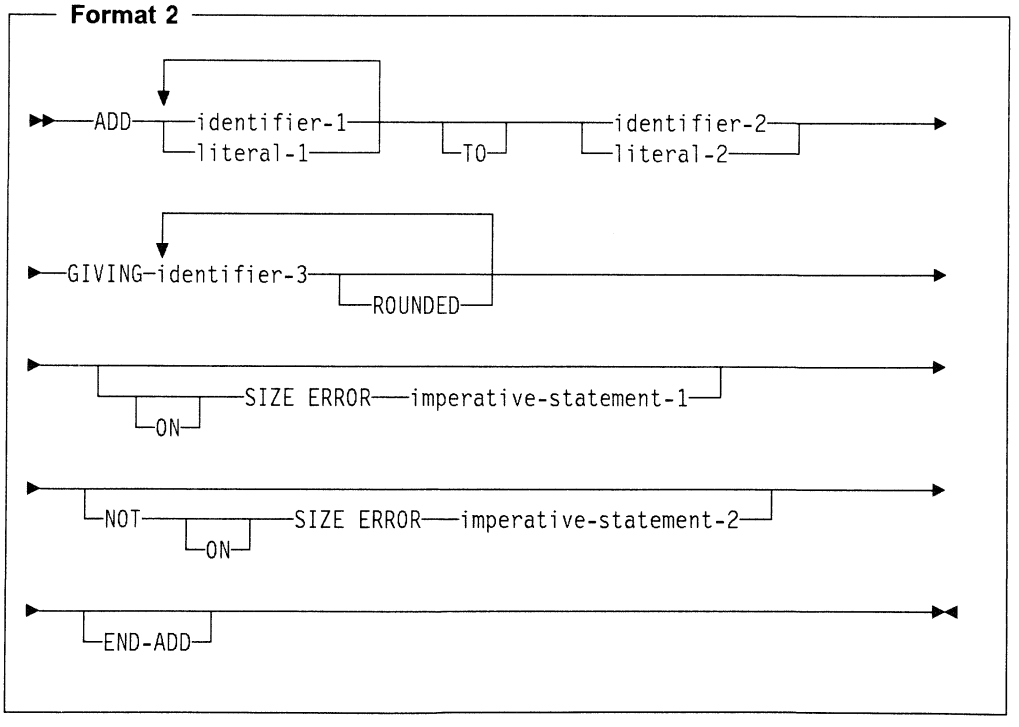
The ADD statement sums two or more numeric operands and stores the result.

Format 1



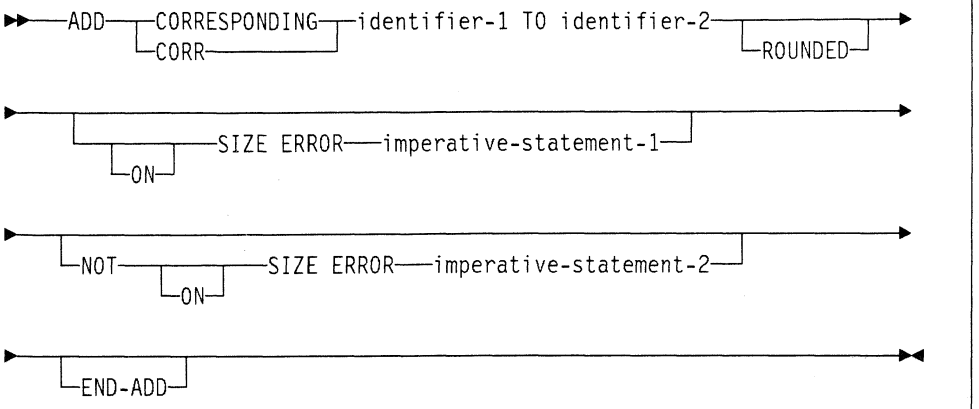
Identifiers and literals preceding the keyword TO are added together, and this initial sum is added to and stored in identifier-2. The initial sum is also added to each successive occurrence of identifier-2, in the left-to-right order in which identifier-2 is specified.

Format 2



The values of the operands preceding the word GIVING are added together, and the sum is stored as the new value of each data item referenced by identifier-3.

Format 3



Elementary data items within identifier-1 are added to and stored in the corresponding elementary items within identifier-2.

For all Formats:

identifier

In Format 1, must name an elementary numeric item.

In Format 2, must name an elementary numeric item, except when following the word GIVING. Each identifier following the word GIVING must name an elementary numeric or numeric-edited item.

In Format 3, must name a group item.

literal

Must be a numeric literal.

If the composite of operands is 18 digits or less, enough places are carried so that no significant digits are lost during execution.

- 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.

IBM Extension

The composite of all operands in an arithmetic statement can have a maximum length of 30 digits.

End of IBM Extension

ROUNDED Phrase

For Formats 1, 2, and 3, see "ROUNDED Phrase" on page 210.

SIZE ERROR Phrases

For Formats 1, 2, and 3, see "SIZE ERROR Phrases" on page 210.

CORRESPONDING Phrase (Format 3)

See "CORRESPONDING Phrase" on page 209.

IBM Extension

Identifiers d1 and/or d2 can be subordinate to a FILLER item.

End of IBM Extension

END-ADD Phrase

This explicit scope terminator serves to delimit the scope of the ADD statement. END-ADD permits a conditional ADD statement to be nested in another conditional statement. END-ADD may also be used with an imperative ADD statement.

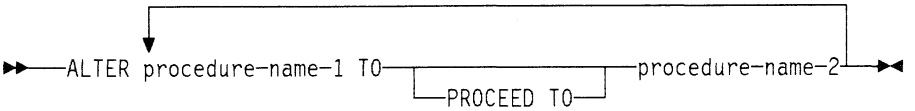
For more information, see "Delimited Scope Statements" on page 207.

ALTER Statement

The ALTER statement changes the transfer point specified in a GO TO statement.

Note: The ALTER statement is an obsolete element and encourages the use of unstructured programming practices; it is to be deleted from the next revision of the ANSI Standard. The EVALUATE statement provides the same function as the ALTER statement and helps to ensure that your program will be well-structured.

Format



The ALTER statement modifies the GO TO statement in the paragraph named by procedure-name-1. Subsequent executions of the modified GO TO statement(s) transfer control to procedure-name-2.

procedure-name-1
Must name a Procedure Division paragraph that contains only one sentence: a GO TO statement without the DEPENDING ON phrase.

procedure-name-2
Must name a Procedure Division section or paragraph.

Before the ALTER statement is executed, when control reaches the paragraph specified in procedure-name-1, the GO TO statement transfers control to the paragraph specified in the GO TO statement. After execution of the ALTER statement, however, the next time control reaches the paragraph specified in procedure-name-1, the GO TO statement transfers control to the paragraph specified in procedure-name-2.

The ALTER statement acts as a program switch, allowing, for example, one sequence of execution during initialization and another sequence during the bulk of file processing. Because altered GO TO statements are difficult to debug, it is preferable to test a switch, and based on the value of the switch, execute a particular code sequence. For example:

```
PARAGRAPH-1.
  GO TO BYPASS-PARAGRAPH.
PARAGRAPH-1A.
    .
    .
BYPASS-PARAGRAPH.
    .
    .
 ALTER PARAGRAPH-1 TO PROCEED TO
     PARAGRAPH-2.
    .
    .
PARAGRAPH-2.
    .
    .
```

Before the ALTER statement is executed, when control reaches PARAGRAPH-1, the GO TO statement transfers control to BYPASS-PARAGRAPH. After execution of the ALTER statement, however, the next time control reaches PARAGRAPH-1, the GO TO statement transfers control to PARAGRAPH-2.

Segmentation Information

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.

Modified GO TO statements in independent segments can sometimes be returned to their initial states. See the *COBOL/400 User's Guide* for a further discussion of segmentation information.

CALL Statement

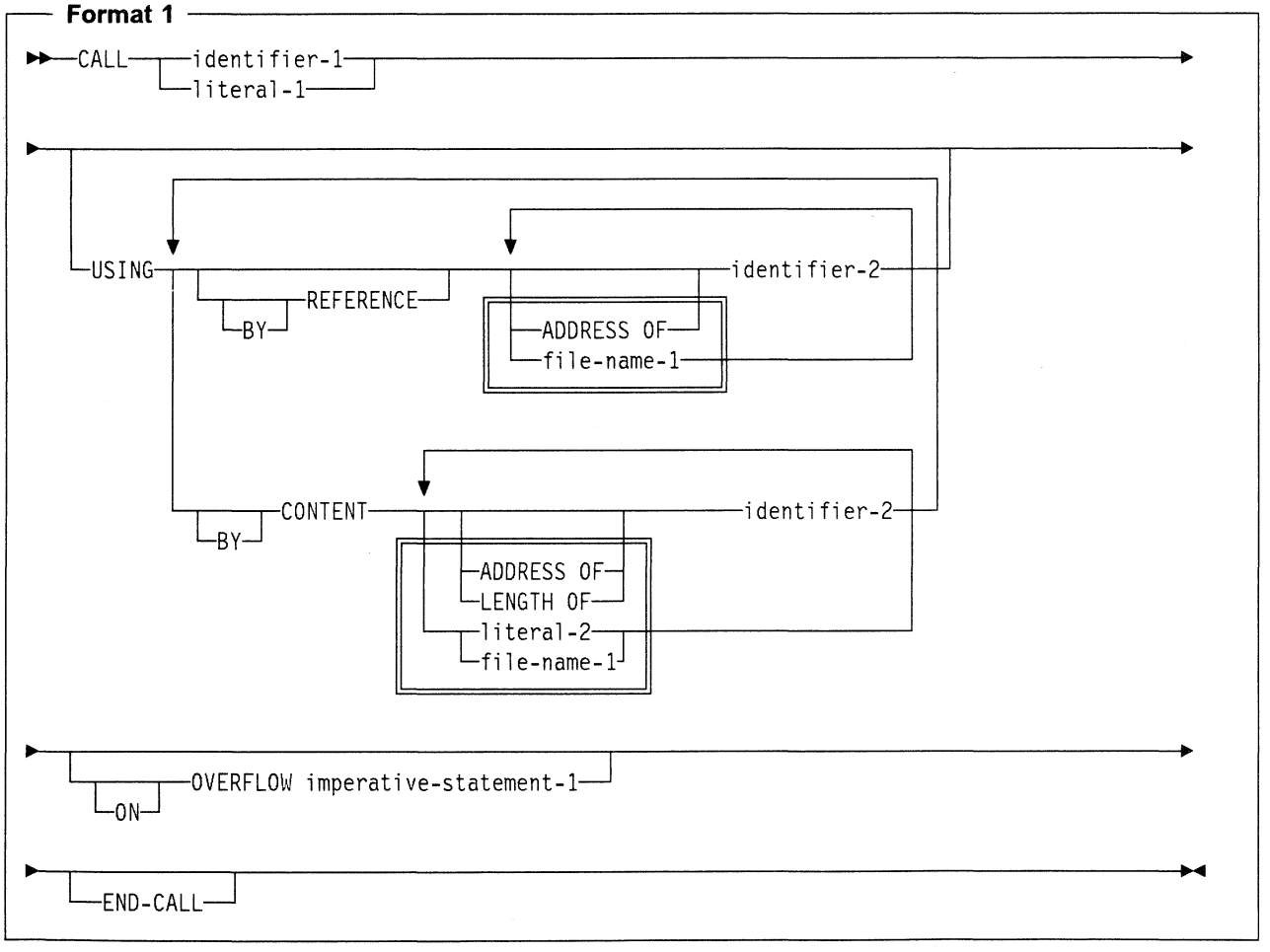
The CALL statement transfers control from one object program to another within the run unit.

The program containing the CALL statement is the calling program; the program identified in the CALL statement is the called subprogram. The calling program must contain a CALL statement at the point where another program is to be called.

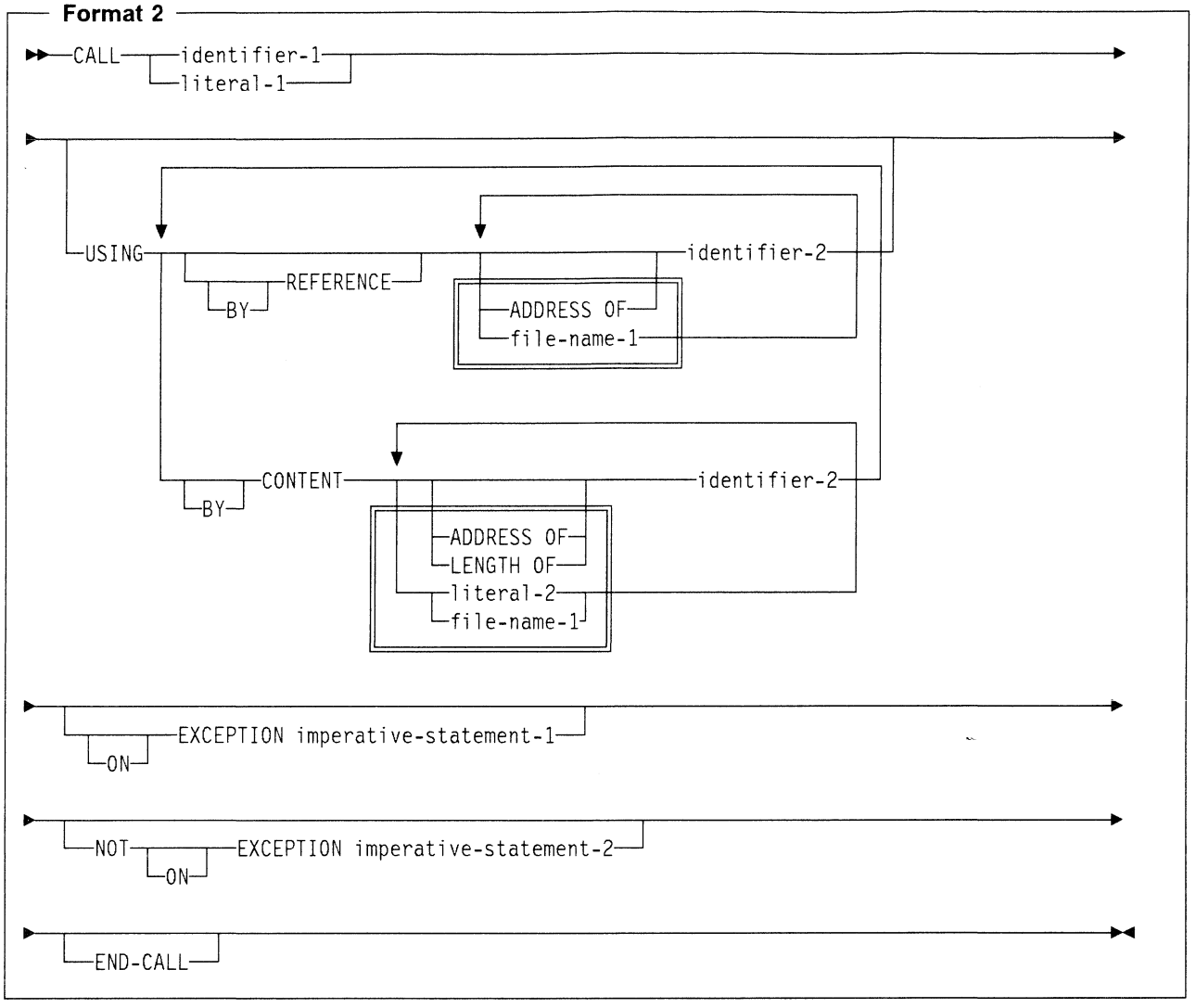
Processing of the CALL statement passes control to the first nondeclarative instruction of the called subprogram. Control returns to the calling program at the instruction following the CALL statement.

Called subprograms themselves can contain CALL statements, but a called subprogram that contains a CALL statement that directly or indirectly calls the calling program gives unpredictable results.

Format 1



Format 2



literal-1

Must be nonnumeric, uppercase, and must conform to the rules for formation of program-names. The first 10 characters of the literal are used to make the correspondence between the calling program and the called subprogram. The literal must specify the program-name of the called subprogram.

If literal-1 is specified, the call is classified as a static call because the PROGRAM-ID is determined at compile time.

identifier-1

Must be an alphanumeric data item. Its contents must conform to the rules for formation of a program-name. The first 10 characters of identifier-1 are used to make the correspondence between the calling and the called program.

If identifier-1 is specified, the call is classified as a dynamic call because the PROGRAM-ID is resolved at run time each time a call is made. If literal-1 is specified, the PROGRAM-ID is resolved only once.

CALL statement processing passes control to the called subprogram, which becomes part of the run unit. If a CALL statement names a program that does not exist in the job's library list (*LIBL) at run time, an error message is issued.

A called subprogram is in its initial state the first time it is called within a run unit. It is also in its initial state the first time it is called after a CANCEL statement. On all other entries into the called subprogram, it is in its last-used state, and the reinitialization of any items is the responsibility of the user.

The user return code is set to 0 at the start of the processing of any COBOL program, and before a call is made to another program. See the RTVJOBA and DSPJOB commands in the *CL Programmer's Guide* for more information about return codes.

USING

Included in the CALL statement only if there is a USING phrase in the Procedure Division header of the called subprogram. USING phrases can contain up to 30 operands, and the number of operands in each must be identical.

For more information about the USING phrase, see "The USING Phrase" on page 183.

COBOL/400 programs can pass arguments to other AS/400 programs. The attributes of the data passed depend on the requirements of the called subprogram. If a called program requires several parameters, you must specify the identity of each parameter, rather than a group item that consists of the parameters.

The sequence of identifiers in the USING phrase of the CALL statement and in the corresponding USING phrase in the called subprogram's Procedure Division header determines the correspondence between the identifiers used by the calling and called programs. This correspondence is by position, rather than by name.

The values of the parameters referenced in the USING phrase of the CALL statement are made available to the called subprogram at the time the CALL statement is executed.

BY REFERENCE Phrase

The value of a parameter passed through the BY REFERENCE phrase is evaluated when the CALL statement runs. This value is assigned to the corresponding parameter of the called program. The number of characters in each parameter must be equal; however, the data descriptions need not be the same.

When a COBOL/400 parameter is passed BY REFERENCE, a pointer to the original data item passes to the called program. Because of this, a change to a parameter in a called program will result in a change to a data item in a calling program.

identifier-2

Must be defined as a level-01, level-77, or elementary data item in the File Section, Working-Storage Section, or Linkage Section.

IBM Extension

It can be:

- A data item of any level in the Data Division
- A pointer data item (an item defined implicitly or explicitly as USAGE IS POINTER).

ADDRESS OF special register
For information about this special register, see page 101. Note that the ADDRESS OF a data item is not allowed in this case.

file-name-1
Must appear in an FD entry. It passes a pointer data item that refers to a File Information Block (FIB).

End of IBM Extension

BY CONTENT Phrase

The value of a parameter passed through the BY CONTENT phrase is evaluated when the CALL statement runs. The value is assigned to the corresponding parameter of the called program.

For each COBOL/400 item passed BY CONTENT, a copy of the item is made in the calling program, and a pointer to this copy passes to the called program. Changes made to the parameter in the called program do not affect the data item of the calling program. The number of characters in each parameter must be equal; however, the data descriptions need not be the same.

identifier-2
Must be defined as a level-01, level-77, or elementary data item in the File Section, Working-Storage Section, or Linkage Section.

IBM Extension

It can be:

- A data item of any level in the Data Division
- A pointer data item (an item defined implicitly or explicitly as USAGE IS POINTER).

ADDRESS OF special register
For information about this special register, see page 101.

ADDRESS OF a data item
For information about this, see page 101.

LENGTH OF special register
For information about this special register, see page 23.

literal-2
Can be:

- A nonnumeric literal
- A figurative constant
- A Boolean literal.

file-name-1
Must appear in an FD entry. It passes a pointer data item that refers to a File Information Block (FIB).

End of IBM Extension

The BY CONTENT and BY REFERENCE phrases apply to the parameters that follow them until another BY CONTENT or BY REFERENCE phrase is encountered. In the absence of both phrases, BY REFERENCE is assumed.

ON EXCEPTION Phrase

An exception condition occurs when the called subprogram cannot be made available. At that time, one of the following occurs:

1. If the ON EXCEPTION phrase appears in the CALL statement, control transfers to imperative-statement-1. Processing then continues according to the rules for each statement specified in imperative-statement-1.

   If a procedure-branching or conditional statement causing explicit transfer of control runs, control transfers according to the rules for that statement. Otherwise, once imperative-statement-1 has run, control transfers to the end of the CALL statement, and the NOT ON EXCEPTION phrase, if specified, is ignored.

2. If the ON EXCEPTION phrase does not appear in the CALL statement, the NOT ON EXCEPTION phrase, if specified, is ignored.

This phrase handles only the exceptions that result from program existence, authority, and storage.

NOT ON EXCEPTION Phrase

If an exception condition does not occur (in other words, the called subprogram can be made available), control transfers to the called program. After control returns from the called program, the ON EXCEPTION phrase, if specified, is ignored, and control transfers to the end of the CALL statement (or to imperative-statement-2, if the NOT ON EXCEPTION phrase is specified).

If control transfers to imperative-statement-2, processing continues according to the rules for each statement specified in imperative-statement-2.

If a procedure-branching or conditional statement causing explicit transfer of control runs, control transfers according to the rules for that statement. Otherwise, once imperative-statement-2 has run, control transfers to the end of the CALL statement.

If you specify this phrase in conjunction with the ON OVERFLOW phrase, an error will result.

ON OVERFLOW Phrase

The ON OVERFLOW phrase has the same effect as the ON EXCEPTION phrase.

END-CALL Phrase

This explicit scope terminator delimits the scope of the CALL statement.

END-CALL permits a conditional CALL statement to be nested in another conditional statement. END-CALL can also be used with an imperative CALL statement.

For more information, see "Delimited Scope Statements" on page 207.

Program Termination Statements

A main program is the highest-level COBOL program invoked in a run unit. A subprogram is a COBOL program invoked by another COBOL program.

Table 27 on page 256 shows the action taken for each program termination statement in both a main program and a subprogram.

Table 27. Program Termination Actions

Termination Statement	Main Program	Subprogram
EXIT PROGRAM	Nonoperational	Return to calling program.
STOP RUN	Return to calling program.[1] (May be the system and cause job to end.[1a])	Return directly to the program that called the main program.[1] (May be the system and cause job to end.[1a])
GOBACK[2]	Return to calling program.[1] (May be the system and cause job to end.[1a])	Return to calling program.

[1] If the main program is called by a program written in another language that does not follow COBOL linkage conventions, return will be to this calling program.

[1a] Execution of the run unit ceases and control is transferred to the operating system. See the chapter on Interprogram Communication in the *COBOL/400 User's Guide* for information about run units.

[2] The GOBACK statement is an IBM extension.

USING Phrase Example

Calling Program Description (PGMA)

```
WORKING-STORAGE SECTION.
01 PARAM-LIST.
   05 PARTCODE PIC A.
   05 PARTNO PIC X(4).
   05 U-SALES PIC 9(5).
     .
     .
     .
PROCEDURE DIVISION.
     .
     .
     .
     CALL PGMB
      USING PARAM-LIST.
```

Called Program Description (PGMB)

```
LINKAGE SECTION.
01 USING-LIST.
   10 PART-ID PIC X(5).
   10 SALES PIC 9(5).
     .
     .
     .
PROCEDURE DIVISION USING
USING-LIST.
```

Note: In the calling program, the code for parts (PARTCODE) and the part number (PARTNO) are referred to separately. In the called subprogram, the code for parts and the part number are combined into one data item (PART-ID); therefore in the called subprogram, a reference to PART-ID is the only valid reference to them.

CALL Statement Considerations

Call by Identifier

A system pointer that associates an identifier with a specific program object is set when you first use the identifier in a CALL statement. The value of this pointer never changes, unless:

- You use a CANCEL statement against the identifier, or,
- You change the value of the identifier and perform a call using this new value.

The CANCEL statement nulls the pointer; when you next use the identifier in a CALL statement, the system pointer is set again. You must run the CANCEL statement from the program that originally performed the call.

Important for Compatibility!

If you carry out a call by an identifier to a program that you subsequently delete or rename, you must use the CANCEL statement to null the system pointer associated with the identifier.

Length of Parameters

If the length of a list of arguments (in bytes) does not match the expected length, unexpected results could occur in the called or calling program.

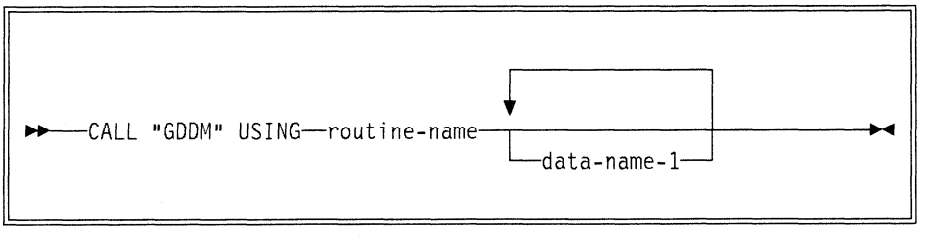
OS/400 Graphics Support

You can use the CALL statement to access the following OS/400 graphics routines:

- Graphical Data Display Manager (GDDM), a set of graphics primitives for drawing pictures
- Presentation Graphics Routines (PGR), a set of business charting routines.

You access all these graphics routines with the same format of the CALL statement:

CALL GDDM



Routine-name is the name of the graphics routine you want to use.

The data-names that follow routine-name are the parameters necessary to use certain graphics routines. The number of parameters that you must specify varies, depending on which routine you select. When you select a graphics

routine, make sure each parameter is the correct size and data type as required by that routine.

The following are examples of calling graphics routines. Remember, you must use the CALL literal format and define each parameter as required by the graphics routine you use.

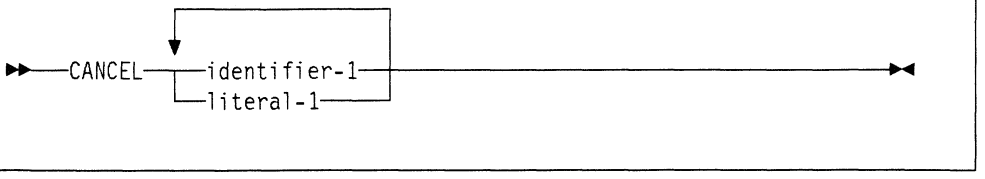
```
MOVE "FSINIT" TO OS-400-GRAPHICS-ROUTINE-NAME.
CALL "GDDM" USING OS-400-GRAPHICS-ROUTINE-NAME.
    .
    .
MOVE "GSFLD" TO OS-400-GRAPHICS-ROUTINE-NAME.
CALL "GDDM" USING OS-400-GRAPHICS-ROUTINE-NAME,
                  PIC-ROW, PIC-COL,
                  PIC-DEPTH, PIC-WIDTH.
```

For more information about graphics routines and their parameters, see the *GDDM Programming Guide* and the *GDDM Programming Reference*.

CANCEL Statement

The CANCEL statement ensures that the next time the referenced subprogram is called it will be entered in its initial state.

Format



literal-1, identifier-1

The name of the subprogram to be canceled. It must be nonnumeric. The contents must conform to the rules for formation of a program-name (see "PROGRAM-ID Paragraph" on page 50).

The first 10 characters of the literal or of the contents of the identifier identify the connection between the calling program and the called subprogram.

Each literal or contents of the identifier specified in the CANCEL statement must be the same as the literal or contents of the identifier specified in an associated CALL statement.

After a CANCEL statement for a called subprogram has been executed, that subprogram no longer has a logical connection to the program. If a CALL statement is executed later by any program in the run unit naming the same subprogram, that subprogram will be entered in its initial state.

A CANCEL statement closes all open files.

You can cancel a called subprogram by either referencing it as the operand of a CANCEL statement or by terminating the run unit of which the subprogram is a member.

A CANCEL statement operates only on the program specified, and not on any program that may have been called by the canceled program.

No action is taken when a CANCEL statement is executed, naming a program that has not been called in the run unit, or that names a program that was called and subsequently canceled. In either case, control passes to the next statement.

If a CANCEL statement names a program that does not exist in the library list, an error message is issued.

Called subprograms may contain CANCEL statements. However, recursive calls are not allowed; a called subprogram must not contain a CANCEL statement that directly or indirectly cancels the calling program itself, or any other program higher than itself in the calling hierarchy. In this case, control is passed to the next statement.

A program named in a CANCEL statement must not refer to any program that has been called and has not yet returned control to the calling program. A program may, however, cancel a program that it did not call, providing that, in the calling hierarchy, it is higher than or equal to the program it is canceling. For example:

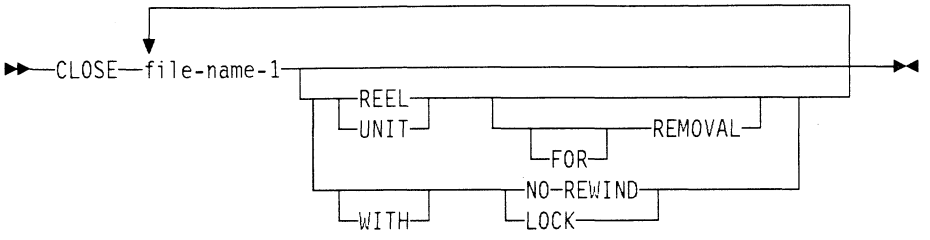
```
A calls B and B calls C        (When A receives control,
                                it can cancel C.)

A calls B and A calls C        (When C receives control,
                                it can cancel B.)
```

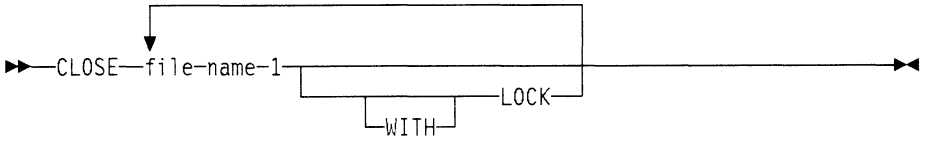
CLOSE Statement

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

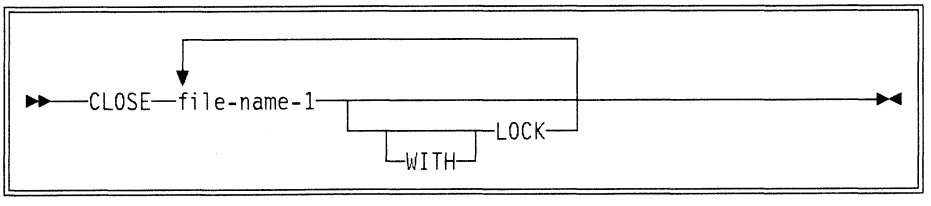
Format 1—Sequential



Format 2—Indexed and Relative



Format 3—TRANSACTION



file-name-1

Designates the file upon which the CLOSE statement is to operate. If more than one file-name is specified, the files need not have the same organization or access. File-name-1 must not be a sort or merge file.

A CLOSE statement may be executed only for a file in an open mode. After successful execution of a CLOSE statement without the REEL/UNIT phrase:

- The record area associated with the file-name is no longer available. Unsuccessful execution of a CLOSE statement leaves availability of the record data undefined.
- An OPEN statement for the file must be executed before any other input/output statement.

If the FILE STATUS clause is specified in the FILE-CONTROL entry, the associated status key is updated when the CLOSE statement is executed. For more information about the status key, see "Common Processing Facilities" on page 214.

If the file is in an open status and the execution of a CLOSE statement is unsuccessful, the EXCEPTION/ERROR procedure (if specified) for this file is executed.

If a CLOSE statement for an open file is not processed before a STOP RUN for this run unit, the file is implicitly closed.

If the SELECT OPTIONAL clause is specified in the file-control entry for this file and the file is not present at run time, standard end-of-file processing is not performed.

CLOSE Statement Considerations

The following tables illustrate organization, access, device, and volume considerations for the CLOSE statement. The letter codes used in the tables are defined in the section following the tables.

Table 28. Sequential Organization

ACCESS	**SEQUENTIAL**				
DEVICE	**PRINTER**	**DISKETTE**	**DISK**	**DATABASE**	**FORMATFILE**
CLOSE	J,K	J,K	J,K	J,K	J,K
CLOSE WITH LOCK	E,J,K	E,J,K	E,J,K	E,J,K	E,J,K
REEL/UNIT	—	—	—	—	—
REMOVAL	—	—	—	—	—
NO REWIND	—	—	—	—	—

Table 29. Sequential Organization

ACCESS	**SEQUENTIAL**	
DEVICE	**TAPEFILE**	
VOLUME	**SINGLE**	**MULTI**
CLOSE	G,J,K,L	A,G,J,K,L
CLOSE WITH LOCK	E,G,J,K,L	A,E,G,J,K,L
CLOSE NO REWIND	B,J,K,L	A,B,J,K,L
CLOSE REEL/UNIT	C,L	F,G,K,L
CLOSE REEL/UNIT FOR REMOVAL	C,L	D,F,G,K,L

Table 30. Indexed Organization		
ACCESS	**ANY**	
DEVICE	**DISK**	**DATABASE**
CLOSE	J,K	J,K
CLOSE WITH LOCK	E,J,K	E,J,K
REEL/UNIT	—	—
REMOVAL	—	—
NO REWIND	—	—

Table 31. Relative Organization		
ACCESS	**ANY**	
DEVICE	**DISK**	**DATABASE**
CLOSE	J,K,M	J,K,M
CLOSE WITH LOCK	E,J,K,M	E,J,K,M
REEL/UNIT	—	—
REMOVAL	—	—
NO REWIND	—	—

Letter Code **Meaning**

— An invalid combination.

A No effect on any previous volumes. Any additional volumes are not processed.

B The current volume is left in its present position. The reel is not rewound.

IBM Extension

The system always rewinds and unloads the tape when REEL/UNIT is specified on the CLOSE statement.

End of IBM Extension

C Optional, but only syntax-checked (performs no function at run time).

D The current volume is rewound and unloaded. The system is notified that the volume is logically removed from this run unit. However, the volume can be accessed again, after processing of a CLOSE statement without the REEL/UNIT phrase and an OPEN statement for this file.

E COBOL ensures that this file cannot be reopened during this processing of the program.

For information about file and record locking, see the *COBOL/400 User's Guide*.

F Close volume procedures. The labels are handled as follows:

MODE OF FILE	LABEL RECORDS	
	STANDARD	OMITTED
Input	F01	F02
Output	F03	F04
I-O	F01	F02

F01 The current volume is positioned to read the labels. If this is the last volume for the file, the next processed READ statement receives the AT END condition. If this is not the last volume, the following actions are taken:

1. The current volume is unloaded
2. The beginning standard labels on the next volume are read
3. The next processed READ statement gets the first record on the newly mounted volume.

F02 The current volume is unloaded. If all of the reels as specified on the REELS parameter of the Create Tape File (CRTTAPF) or Override with Tape File (OVRTAPF) CL command have been processed, the next processed READ statement receives the AT END condition. If there are more reels, the next volume is mounted, and the next processed READ statement gets the first record on the newly mounted volume.

F03 The standard end-of-volume labels are written. The next volume is mounted. The standard beginning labels are written on the new volume. The next processed WRITE statement places the next logical record on the newly mounted volume.

F04 The system end-of-volume procedures for nonlabeled tapes are run. The next volume is mounted. The system beginning of volume procedures for nonlabeled tapes are run. The next processed WRITE statement places the next logical record on the newly mounted volume.

G The current volume is positioned at its beginning.

J The record area associated with the file-name is no longer available after successful processing of this statement. Unsuccessful processing of this statement leaves availability of the record data area undefined.

Labels are processed as follows:

MODE OF FILE	LABEL RECORDS	
	STANDARD	OMITTED
Input	J01	J02
Output	J03	J04
I-O	J01	J02

J01 If the file is positioned at its end, the label records are read and verified, and the file is closed. If the file is not at its end, the file is closed.

J02 The file is closed.

J03 The standard label records are written, and the file is closed.

J04 The file is closed without any label processing.

K May be processed only for an open file.

L If the ENDOPT keyword has been specified in the Create Tape File (CRTTAPF) or the Override with Tape File (OVRTAPF) CL command, the positioning of the tape volume when the file is closed follows the value specified in that keyword (a value of LEAVE, REWIND, or UNLOAD). If the ENDOPT keyword has not been specified, the positioning of the tape volume when the file is closed corresponds to the following ENDOPT values:

```
CLOSE NO REWIND statement . . . .  LEAVE
All other forms of CLOSE  . . . .  REWIND
```

M To extend a relative file boundary beyond the current number of records and within the file size, use the INZPFM command to add deleted records before processing the file. You will need to do this when more records need to be added to the file, and file status 0Q has been received.

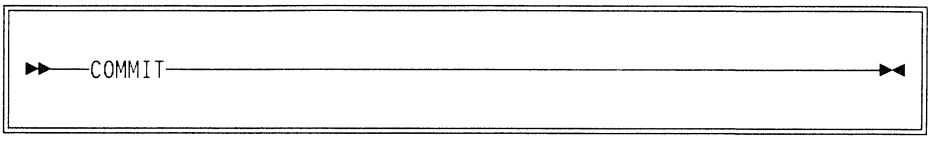
Any attempt to extend a relative file beyond its current size results in a boundary violation.

IBM Extension

COMMIT Statement

The COMMIT statement provides a way of synchronizing changes to data base records while preventing other jobs from modifying those records until the COMMIT is performed. The format of the COMMIT statement is:

COMMIT Statement



When the COMMIT statement is executed, all changes made to files under commitment control since the previous commitment boundary are made permanent. A commitment boundary is established by the successful execution of a ROLLBACK or COMMIT statement. If no COMMIT or ROLLBACK has been issued in the current job, a commitment boundary is established by the first OPEN of any file under commitment control in the job. Changes are made to all files under commitment control in the job, not just to files under commitment control in the COBOL program that issues the COMMIT statement.

When a COMMIT is executed, all record locks held by the job since the last commitment boundary for files under commitment control are released and the records become available to other jobs.

The COMMIT statement only affects files under commitment control. If a COMMIT is executed and there are no files opened under commitment control, the COMMIT statement has no effect and no commitment boundary is established.

The COMMIT statement does *not*:

- Modify the I-O-FEEDBACK area for any file
- Change the file position indicator for any file
- Set a file status value for any file.

For more information on commitment control, see the *COBOL/400 User's Guide*.

End of IBM Extension

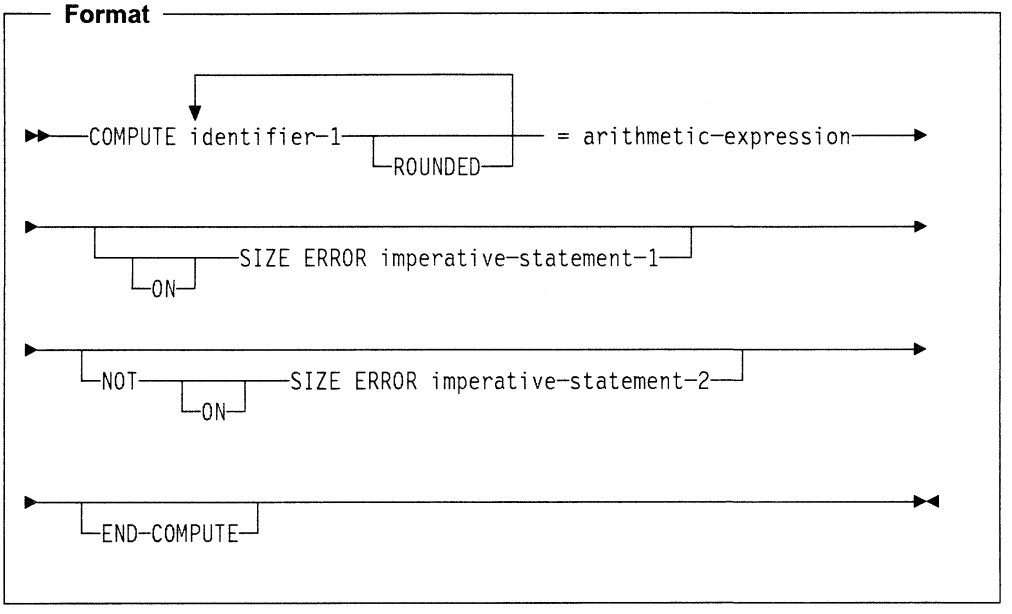
COMPUTE Statement

The COMPUTE statement assigns the value of an arithmetic expression to one or more data items.

With the COMPUTE statement, arithmetic operations can be combined without the restrictions on receiving data items imposed by the rules for the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements.

When arithmetic operations are combined, the COMPUTE statement may be more efficient than the separate arithmetic statements written in series.

Format



identifier-1

Must name either elementary numeric item(s) or elementary numeric-edited item(s).

arithmetic-expression

May be any arithmetic expression, as defined in "Arithmetic Expressions" on page 187.

When the COMPUTE statement is executed, the value of the arithmetic expression is calculated, and this value is stored as the new value of each data item referenced by identifier-1.

An arithmetic expression consisting of a single identifier or literal allows the user to set the value of the data item(s) referenced by identifier-1 equal to the value of that identifier or literal.

ROUNDED Phrase

For a discussion of the ROUNDED phrase, see "ROUNDED Phrase" on page 210.

SIZE ERROR Phrases

For a discussion of the SIZE ERROR phrases, see "SIZE ERROR Phrases" on page 210.

END-COMPUTE Phrase

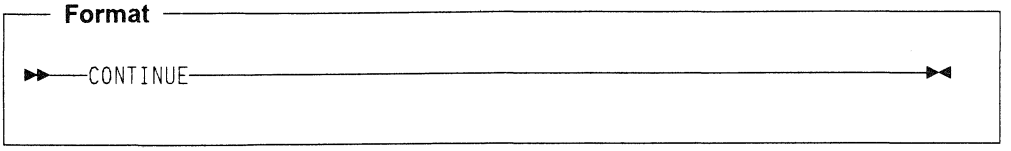
This explicit scope terminator serves to delimit the scope of the COMPUTE statement. END-COMPUTE permits a conditional COMPUTE statement to be nested in another conditional statement. END-COMPUTE may also be used with an imperative COMPUTE statement.

For more information, see "Delimited Scope Statements" on page 207.

CONTINUE Statement

The CONTINUE statement allows you to specify a no operation statement. CONTINUE indicates that no executable instruction is present.

Format



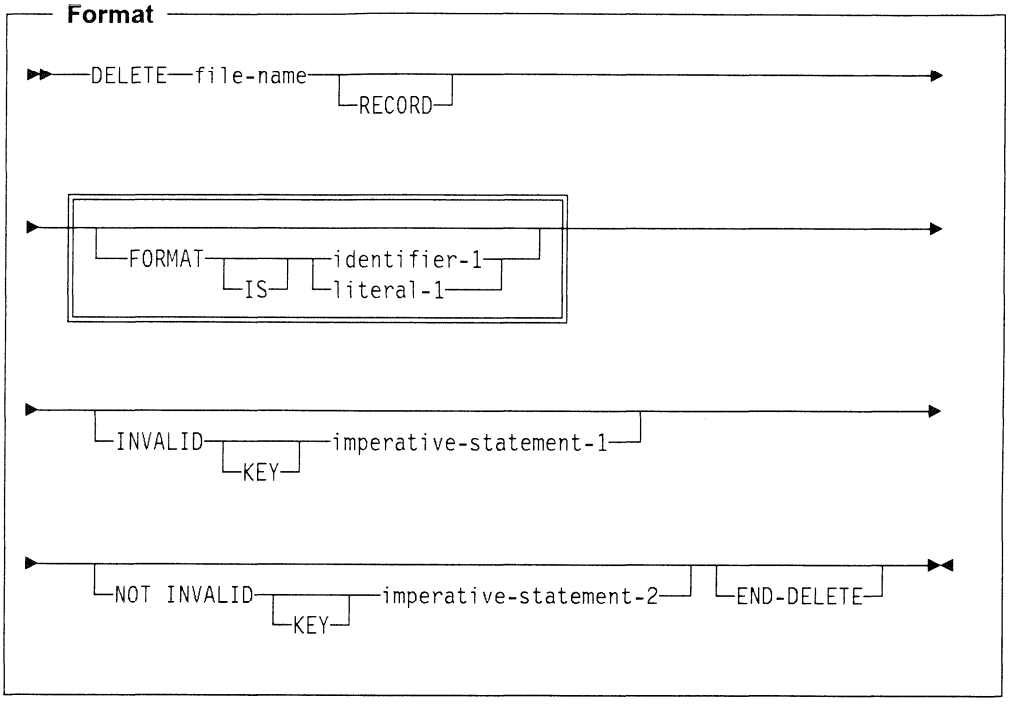
The CONTINUE statement may be used anywhere a conditional statement or an imperative statement may be used. It has no effect on the execution of the program.

DELETE Statement

The DELETE statement removes a record from an indexed or relative file. For indexed files, the key may then be reused for record addition. For relative files, the space is then available for a new record with the same RELATIVE KEY value.

When the DELETE statement is executed, the associated file must be open in I-O mode.

Format



file-name
Must be defined in an FD entry in the Data Division and must be the name of an indexed or relative file.

After successful execution of a DELETE statement, the record is logically removed from the file and can no longer be accessed. Execution of the DELETE statement does not affect the contents of the record area associated with the file-name.

If the FILE STATUS clause is specified in the File-Control entry, the associated status key is updated when the DELETE statement is executed.

The file position indicator is not affected by the processing of the DELETE statement.

DELETE Statement Considerations

The following tables illustrate organization, access, and device considerations for the DELETE statement. The letter codes used in the tables are defined in the section following the tables.

Table 32. Sequential Organization	
Device	**Any**
Access	**SEQUENTIAL**
DELETE Verb	Not Allowed

Table 33. Relative Organization						
Device	**DISK**			**DATABASE**		
Access	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**
DELETE Verb	A,I,P,Z	B,P,Z	B,P,Z	A,I,P,Z	B,P,Z	B,P,Z
INVALID KEY	–	O,U	O,U	–	O,U	O,U
NOT INVALID KEY	–	O,V	O,V	–	O,V	O,V
FORMAT	–	–	–	–	–	–

Table 34. Indexed Organization						
Device	**DISK**			**DATABASE**		
Access	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**
DELETE Verb	A,I,P,Z	D,P,Z	D,P,Z	A,I,P,Z	D,P,Z	D,P,Z
INVALID KEY	–	O,U	O,U	–	O,U	O,U
NOT INVALID KEY	–	O,V	O,V	–	O,V	O,V
FORMAT	–	–	–	–	S,F	S,F

Letter Code	Meaning
–	Combination is not valid.
A	When the DELETE statement is processed, the system logically removes the record retrieved by a READ statement. The last input/output statement must have been a successfully processed READ statement **without** the NO LOCK phrase.

IBM Extension

If the last input/output statement was a successfully processed READ statement **with** the NO LOCK phrase:

- The file status key, if defined, is set to 9S.
- The EXCEPTION/ERROR procedure, if any, is run.
- The DELETE statement is not processed.

End of IBM Extension

If the last input/output statement was not a successfully processed READ statement, the file status key (if defined) is set to 43.

See the *COBOL/400 User's Guide* for information about file and record locking.

B The system logically removes the record identified by the contents of the RELATIVE KEY data item. If the file does not contain such a record, an INVALID KEY condition exists. The space is then available for a new record with the same RELATIVE KEY value.

D The system logically removes the record identified by the contents of the RECORD KEY data item. If the file does not contain such a record, an INVALID KEY condition exists.

IBM Extension

When EXTERNALLY-DESCRIBED-KEY is specified for the file, the key field in the record area[1] for the format specified by the FORMAT phrase is used to find the record to be deleted. If the FORMAT phrase is not specified, the first format defined in the program for the file is used to find the record to be deleted.

DUPLICATES Phrase

If this phrase was specified for the file, the last input/output statement processed for this file before the processing of the DELETE statement must have been a successfully processed READ statement **without** the NO LOCK phrase. The record read by that statement is the record that is deleted.

In this case, the FORMAT phrase is not used to find the record to be deleted. The READ statement is required to ensure that the proper record is deleted when there are duplicates.

If a successful READ operation did not occur before the delete operation:

- The file status key, if defined, is set to 94.
- The EXCEPTION/ERROR procedure, if any, is run.
- The DELETE statement is not processed.

If the last input/output statement was a successfully processed READ statement **with** the NO LOCK phrase:

- The file status key, if defined, is set to 9S.
- The EXCEPTION/ERROR procedure, if any, is run.
- The DELETE statement is not processed.

The value of the RECORD KEY data item should not have changed since the record was read. By default, if it has changed:

- The file status key, if defined, is set to 21.
- An INVALID KEY condition exists.
- The DELETE operation does not occur.

If the *NOFS21DUPKY option is in effect, the value of the RECORD KEY data item can be different. Note that this option is provided for compatibility only; its use is not recommended.

End of IBM Extension

[1] The key field in the record area is the location in the buffer selected in accordance with a record format or specification in order to build a search argument.

IBM Extension

F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format must be operated on.

If an identifier is specified, it must be a character-string of ten characters or less, and it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

If a literal is specified, it must be an uppercase character-string of ten characters or less.

A value of all blanks is treated as though the FORMAT phrase were not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is invoked, if applicable for the file.

End of IBM Extension

I For a file in sequential access mode, the INVALID KEY and NOT INVALID KEY phrases must **not** be specified; however, an EXCEPTION/ERROR procedure may be specified.

For information about error handling, see "Common Processing Facilities" on page 214.

O You can specify this combination.

P Allowed when the file is opened for I-O.

S Required when processing a file that has multiple record formats and has unique keys.

U The INVALID KEY phrase must be specified for files for which an applicable USE procedure is not specified. For more information, refer to "INVALID KEY Condition" on page 215.

V After the processing of a DELETE statement for which there is a NOT INVALID KEY phrase, control transfers to the imperative statement associated with the phrase.

IBM Extension

Z The action of this statement can be inhibited at program run time by the inhibit write (INHWRT) parameter of the Override with database file (OVRDBF) CL command. When this parameter is specified, non-zero file status codes are not set for data dependent errors. Duplicate key and data conversion errors are examples of data dependent errors.

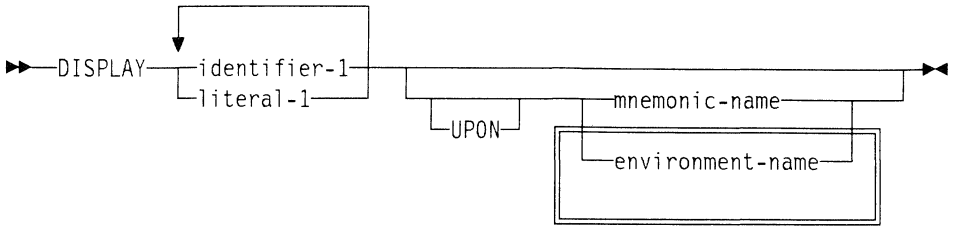
See the *CL Reference* for more information on this command.

End of IBM Extension

DISPLAY Statement

The DISPLAY statement transfers the contents of each operand to the output device. The contents are displayed on the output device in the order, left to right, in which the operands are listed.

Format 1—Data Transfer



identifier-1

If it is numeric and is not described as external decimal, the identifier is converted automatically to external format, as follows:

- Binary or internal decimal items are converted to external decimal. Negative signed values cause a low-order sign to be displayed. For example, if SIGN WITH SEPARATE CHARACTER is not specified and two numeric items have the values -34 and 34, they are displayed as 3M and 34, respectively. If SIGN WITH SEPARATE CHARACTER is specified, a + or a - sign is displayed as either leading or trailing, depending on how the number was specified.

  Note: Group items containing packed or binary data (COMP, COMP-3, PACKED-DECIMAL, BINARY, or COMP-4) should not be displayed on a display station. Such data can contain display station control characters which can cause undesirable and unpredictable results.

- No other identifiers require conversion.

literal-1

May be any figurative constant. When a figurative constant is specified, only a single occurrence of that figurative constant is displayed.

Each numeric literal must be an unsigned integer.

IBM Extension

Signed noninteger numeric literals are allowed.

End of IBM Extension

UPON

mnemonic-name must be associated in the SPECIAL-NAMES paragraph with either the work station (REQUESTOR) or the system operator's message queue (CONSOLE or SYSTEM-CONSOLE).

IBM Extension

environment-name may be specified in place of **mnemonic-name**. Valid environment-names are CONSOLE and SYSOUT.

End of IBM Extension

When the UPON phrase is omitted, the DISPLAY statement sends output to the REQUESTOR.

The DISPLAY statement transfers the data in the sending field to the output device. The size of the sending field is the total character count of all operands listed. 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. If the hardware device is not capable of receiving data of the same size as the data item being transferred, then one of the following applies:

- If the total character count is less than the device maximum logical record size, the remaining rightmost characters are padded with spaces.
- If the total character count exceeds the maximum, as many records are written as are needed to display all operands. Any operand being printed or displayed when the end of a record is reached is continued in the next record.

After the last operand has been transferred to the output device, the device is reset to the leftmost position of the next line of the device.

The logical record length depends on the device as follows:

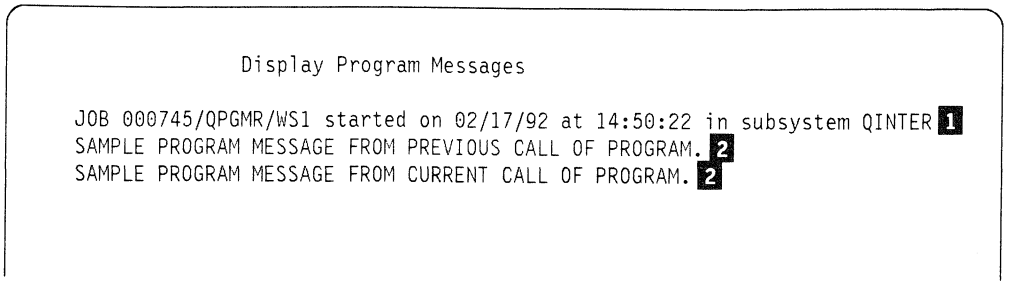
Output	Maximum Logical Record Size
Job log	120 characters
Work station	58 characters
System operator's message queue	58 characters

When a program in a batch job processes a DISPLAY statement without the UPON phrase, or with an UPON phrase associated with the REQUESTOR, the output is sent to the job log in an informational message of severity 99. You can change the severity of this message using the Change Message Description (CHGMSGD) CL command. For more information, see the *CL Reference*.

For an interactive job that uses display device files, DISPLAY statements are not normally used. If you do use them, the following considerations apply.

When an interactive job processes a DISPLAY statement, the logical record appears on the screen in the Program Messages display.

The following screen shows a sample Program Messages display.



1 System messages for this session.

2 Program messages for this session.

This display contains messages from the current program processing, as well as messages relating to other activities in the session.

When a DISPLAY statement is processed, the characteristics of the display device file on the screen determine whether or not to suspend program processing:

RSTDSP(*NO)

If you specify this parameter when you change or create the display device file, DISPLAY statement processing suspends program processing, and the Program Messages display appears on the screen. Press Enter to resume program processing and immediately return the previous display to the screen.

RSTDSP(*YES)

If you specify this parameter when you change or create the display device file, or run the DISPLAY statement from the Command Entry display, DISPLAY statement processing does not suspend program processing.

The Program Messages display appears on the screen and remains there until one of two things happens:

- The program processes a nonsubfile READ or WRITE statement for the file. The Program Messages display then disappears, and the previous display returns to the screen.
- The program ends.

Note: If you want to suspend program processing, code an ACCEPT statement after the DISPLAY statement. This suspends program processing until you press Enter.

To view output records after the program terminates, press the F10 key from the Command Entry display.

For additional information on interactive processing, see the *COBOL/400 User's Guide*. For additional information on the RSTDSP parameter, see the CHGDSPF and CRTDSPF commands in the *CL Reference*.

When a program started by a workstation operator sends a DISPLAY to the system operator's message queue (separate from the workstation), program processing is not suspended.

The location of the output data is dependent upon the type of program initiation as follows:

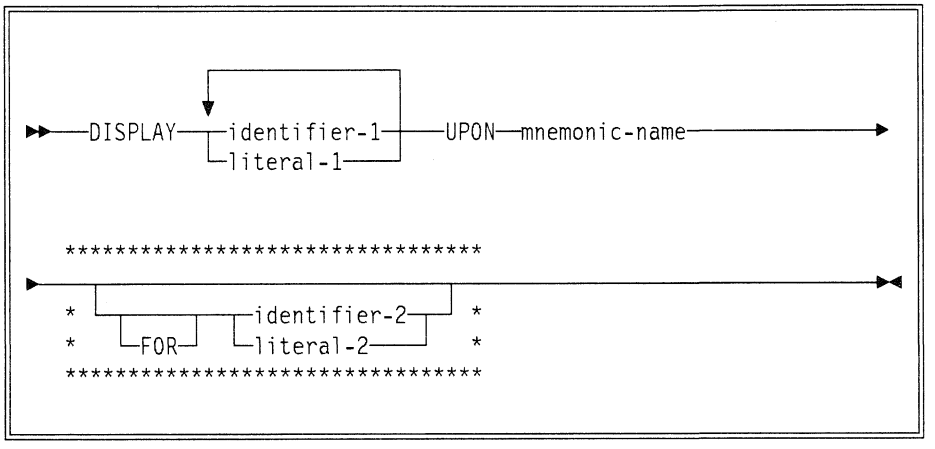
Method of Initiation	Mnemonic-Name Associated with SYSTEM-CONSOLE	Mnemonic-Name Associated with REQUESTOR	UPON Phrase Omitted
BATCH	System operator's message queue	Job log	Job log
INTERACTIVE	System operator's message queue	Work station	Work station

IBM Extension

Local Data Area

This format is used to transfer data to the system-defined local data area created for a job.

Format 2—Local Data Area



This format is only applicable when the mnemonic-name in the SPECIAL-NAMES paragraph is associated with the environment-name LOCAL-DATA.

The DISPLAY statement's literal operands, or the contents of the DISPLAY statement's identifier operands, are written to the system-defined local data area of the job containing the program that issues the DISPLAY. The data is written to the local data area according to the rules of the MOVE statement for a group move, without the CORRESPONDING phrase, and without padding on the right with spaces.

The FOR phrase, when specified, is syntax checked during compilation but is treated as comments during execution. The value of literal-2 or identifier-2 indicates the program device name of the device that is writing data to the local data area. There is only one local data area for each job, and all devices in a job access the same local data area. Literal-2, if specified, must be nonnumeric

and 10 characters or less in length, and identifier-2, if specified, must refer to an alphanumeric data item 10 characters or less in length.

For more information about the local data area, see the *CL Programmer's Guide* and the *COBOL/400 User's Guide*.

Format 3 — Extended DISPLAY Statement

You can use this format of the DISPLAY statement if you specify the *EXTACCDSP generation option of the CRTCBLPGM command, or the EXTACCDSP option of the PROCESS statement. See the *COBOL/400 User's Guide* for information about these options.

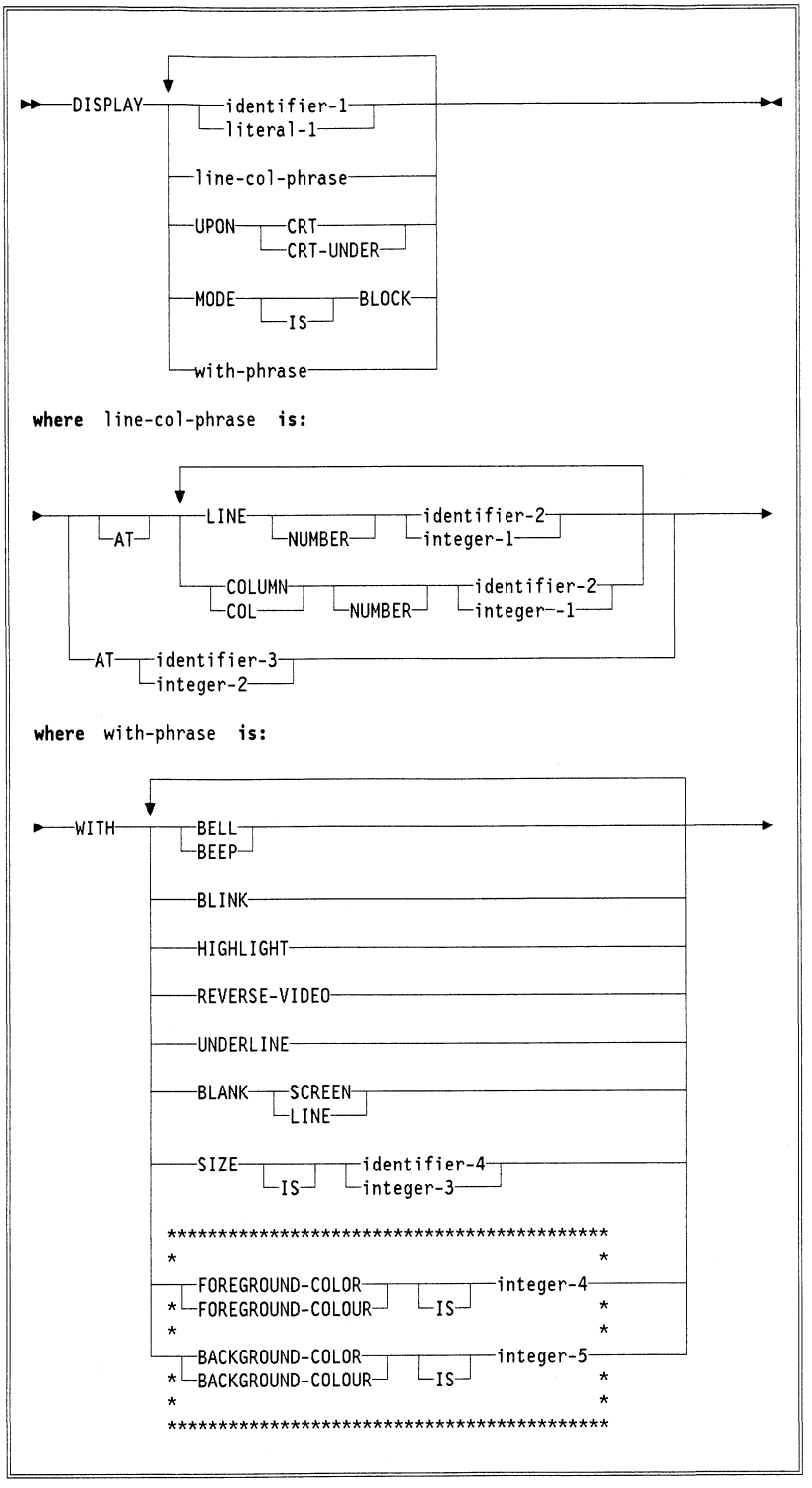
A DISPLAY statement is considered an **extended** DISPLAY statement if it:

- has an AT phrase, or
- has an UPON CRT/CRT-UNDER phrase, or
- has a MODE IS BLOCK phrase, or
- has a WITH phrase, or
- does not have an UPON phrase, but CONSOLE IS CRT is specified in the SPECIAL-NAMES paragraph.

A DISPLAY statement is considered a **standard** DISPLAY statement if it:

- has an UPON phrase (other than UPON CRT/CRT-UNDER), or
- does not have an UPON phrase and CONSOLE IS CRT is not specified in the SPECIAL-NAMES paragraph.

Format 3—Workstation I/O



Part of this statement can be repeated to allow the display of several data items. If the first identifier has no AT, LINE, or COLUMN phrase, it begins in line 1, column 2. Each subsequent data item begins at the currently available screen position following the previous data item.

If identifier-1 or literal-1 is not specified, neither the MODE IS BLOCK phrase nor the WITH phrase is allowed.

If identifier-1 is a group item and there is no MODE IS BLOCK phrase, those elementary subordinate items that have names other than FILLER are displayed. They are displayed simultaneously, and positioned on the screen in the order that their descriptions appear in the DATA DIVISION, separated by the lengths of the FILLER items in the group. For this purpose, the first position on a line is regarded as immediately following the last position on the previous line. When items are separated by FILLERs, the attribute bytes are included in the FILLER length. Thus a FILLER of one or two bytes would contain both the trailing and leading attributes of separate items. In the case of a one-byte FILLER, the trailing and leading attributes would occupy the same byte. Since data items are normally separated by one attribute byte, one-byte FILLERs are not necessary.

If no identifier or literal is present, the DISPLAY operation changes the screen position without actually displaying any data.

The phrases following the identifier or literal can be in any order. All phrases specified apply to the previous identifier or literal, if one was specified. The WITH and MODE phrases cannot be specified if an identifier or literal was not previously specified.

Identifiers or literals in a DISPLAY statement follow one after another, separated by one attribute byte, unless an AT, LINE, or COLUMN phrase is specified. If no AT, LINE, or COLUMN phrase appears in the statement, the first identifier or literal begins at line 1, column 2, followed immediately by all other identifiers or literals.

AT Phrase

The AT phrase indicates the absolute address on the screen at which the DISPLAY operation is to start. It does not indicate the starting position of the leading attribute.

The LINE phrase specifies the line at which the screen item starts on the screen.

The COLUMN phrase specifies the column at which the screen item starts on the screen.

The LINE and COLUMN phrases can appear in any order.

COL is an abbreviation for COLUMN.

identifier-2
integer-1

Identifier-2 and integer-1 must be unsigned numeric integers greater than or equal to zero. If LINE or COLUMN is negative, the absolute value is taken.

Certain combinations of line and column numbers have special meaning:

- Until the column comes within range, out-of-range column values are reduced by the line length, and the line value is incremented. A column number, then, can cause the line number to be incremented several times.
- Out-of-range line values cause the screen to scroll up one line. The effect is the same as if the line number of the bottom line were specified. The screen is never scrolled up by more than one line, regardless of the line specified.

- If column and line numbers are both out of range, out-of-range columns are handled first, followed by out-of-range lines (according to the preceding rules).
- If the line and column numbers given are both zero, the DISPLAY operation starts at the position following the one at which the preceding DISPLAY operation finished. Column 1 of each line is considered to follow the last column of the previous line.
- If the line number is zero, but the column number is not, the DISPLAY operation starts at the specified column on the line following the one at which the preceding DISPLAY operation finished.
- If the column number is zero, but the line number is not, the DISPLAY operation starts on the specified line at the column following the one at which the preceding DISPLAY operation finished.

identifier-3
integer-2

Identifier-3 must be a PIC 9(4) or a PIC 9(6) field. Integer-2 must be a 4- or 6-byte numeric field.

If identifier-3 or integer-2 is 4 digits long, the first two digits specify the line, and the second two digits specify the column. If identifier-3 or integer-2 is 6 digits long, the first three digits specify the line, and the second three digits specify the column.

UPON CRT/CRT-UNDER Phrase

Indicates that the DISPLAY statement is extended.

CRT-UNDER also underlines the displayed data item preceding the UPON CRT-UNDER phrase.

MODE IS BLOCK Phrase

The identifier is treated as an elementary item. Even if it is a group item, it is displayed as one item.

WITH Phrase

The WITH phrase allows the user to specify certain options for the DISPLAY operation. These options are described in the following phrases.

BELL (BEEP) Phrase

An audible alarm sounds each time the item containing this phrase is displayed.

BELL and BEEP can be used interchangeably.

The BLANK Phrase

BLANK is effective each time the screen item containing this clause is displayed.

BLANK LINE erases from the current cursor position to the end of the current line. BLANK SCREEN erases the entire screen and places the cursor at line 1, column 2.

The erasing is done before the item is displayed.

BLINK Phrase
The screen item blinks when it appears on the screen.

HIGHLIGHT Phrase
The screen item is in high-intensity mode when it appears on the screen.

REVERSE-VIDEO Phrase
The screen item is displayed in reverse image.

SIZE Phrase
Specifies the current size of the data item on the screen. You can use this phrase with elementary data items only.

identifier-4
integer-3
Identifier-4 must be an unsigned numeric integer, and must not be subject to an OCCURS clause. Integer-3 must be unsigned.

If identifier-4 or integer-3 has a sign, the compiler uses the absolute value, and issues a warning message.

The SIZE phrase has no effect if the size you specify is zero. In this case, the length of the field is used to display the data item.

If you specify a size that is less than the size implied by the associated PICTURE clause, only the leftmost portion of the data item appears on the workstation display.

For a single elementary data item, the maximum number of characters that you can specify is 2 983 under the UNDSPCHR option, and 2 986 under the NOUNDSPCHR option. If you specify a SIZE literal whose value is too large, a warning message is issued, and the phrase is ignored.

For justified items, only the rightmost portion appears when you specify a size that is smaller than the length of the item.

If the size you specify is greater than the size implied by the PICTURE clause, the displayed version of the item is padded with spaces. The padding occurs on the right for items without the JUSTIFIED phrase, and on the left for justified ones.

ALL figurative constants are displayed as many times as necessary to reach the length you specify. If the display wraps around to a new line, the new line starts at the beginning of the constant.

The following is an example of displaying a figurative constant where the size specified is greater than the figurative constant and wraps around to a new line:

```
DISPLAY ALL 'ABCD' AT 0270 WITH SIZE 15.
```

This constant will be displayed on the screen starting with line 2, column 70:

```
                0000000001              677777777778
                1234567890.............901234567890
Line 1
Line 2                                   ABCDABCDABC
Line 3          ABCD
```

Notice the differences between the following examples:

```
Statement 1 DISPLAY 'WORKSTATION' AT 0275 WITH SIZE 10
Statement 2 DISPLAY ALL 'WORKSTATION' AT 0275 WITH SIZE 10

                0000000001              677777777778
                1234567890.............901234567890
Statement 1                                    WORKST
                ATIO

Statement 2                                    WORKST
                WORK
```

UNDERLINE Phrase

The screen item is underlined when it appears on the screen.

Phrases that are Syntax Checked Only

The following phrases are syntax checked but are treated as documentation by the compiler.

FOREGROUND COLOR or FOREGROUND COLOUR
BACKGROUND COLOR or BACKGROUND COLOUR

Format 3 Considerations

Data items must not contain variable-length tables. A data item can contain a fixed-length table whether or not MODE IS BLOCK has been specified. Fixed-length tables are treated as group items (MODE IS BLOCK is not specified) that are repeated from the first occurrence to the last occurrence of the table.

Some extended DISPLAY statement considerations also apply to the extended ACCEPT statement. (See "Extended ACCEPT and Extended DISPLAY Considerations" on page 241 for more information.)

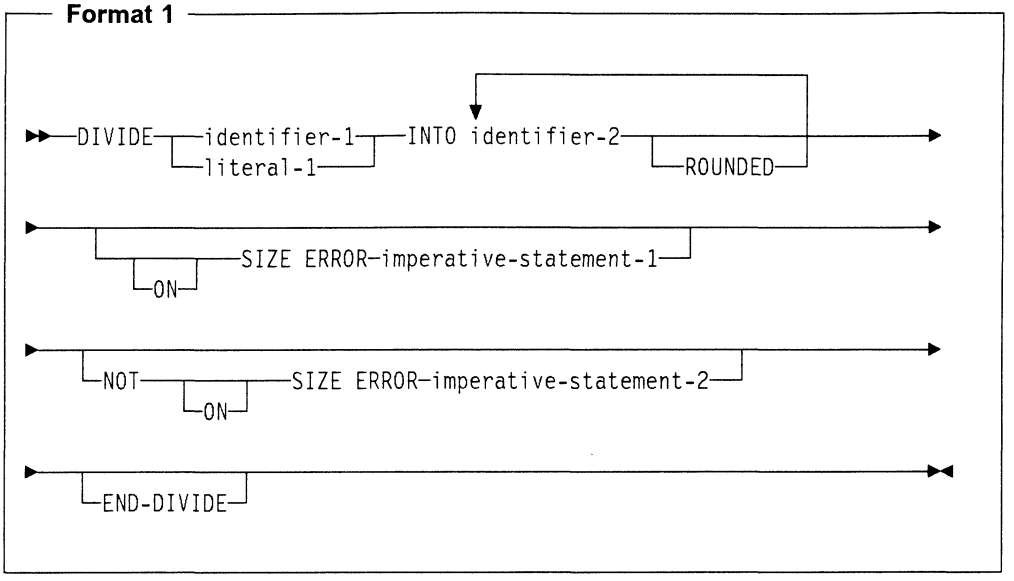
The COBOL/400 extended DISPLAY statement is similar to the IBM COBOL/2 DISPLAY statement (Format 2). The exceptions are discussed in Appendix I, "ACCEPT/DISPLAY and COBOL/2 Considerations" on page 522.

End of IBM Extension

DIVIDE Statement

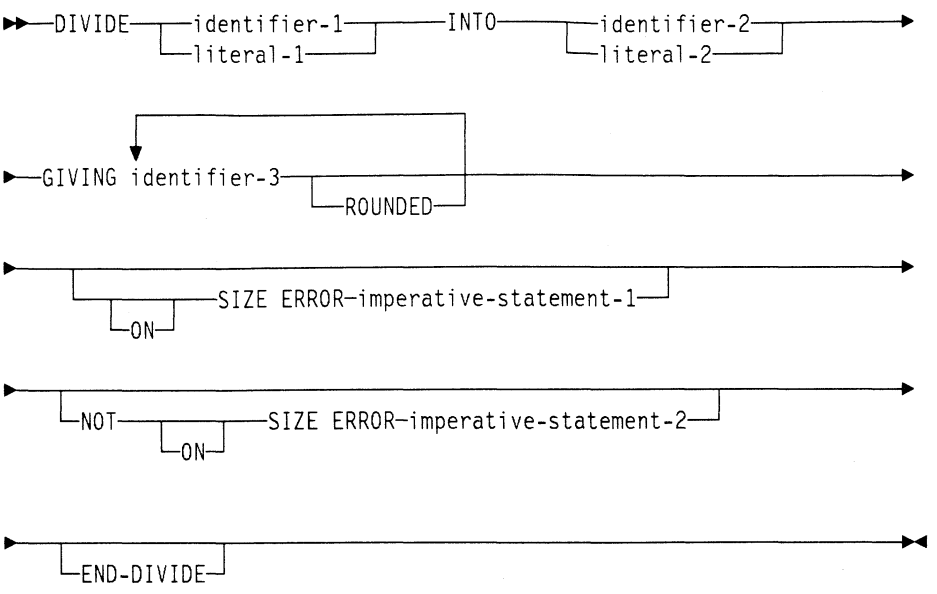
The DIVIDE statement divides one numeric data item into or by one or more others, and sets the values of data items equal to the quotient and remainder.

Format 1

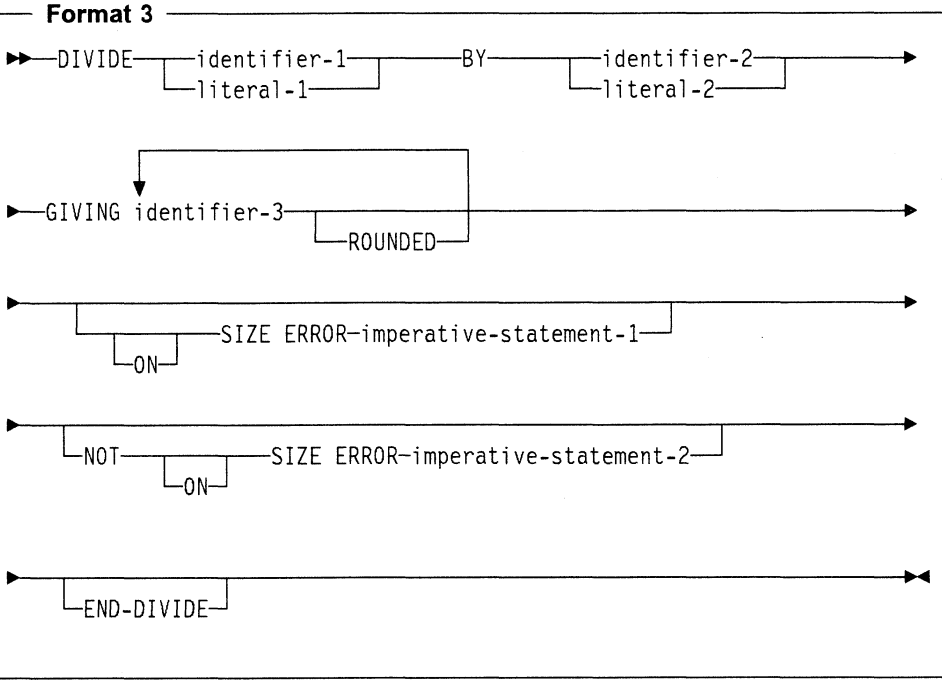


In Format 1, the value of identifier-1 or literal-1 is divided into the value of identifier-2; the quotient is then placed in identifier-2. This process is repeated for each successive occurrence of identifier-2.

Format 2

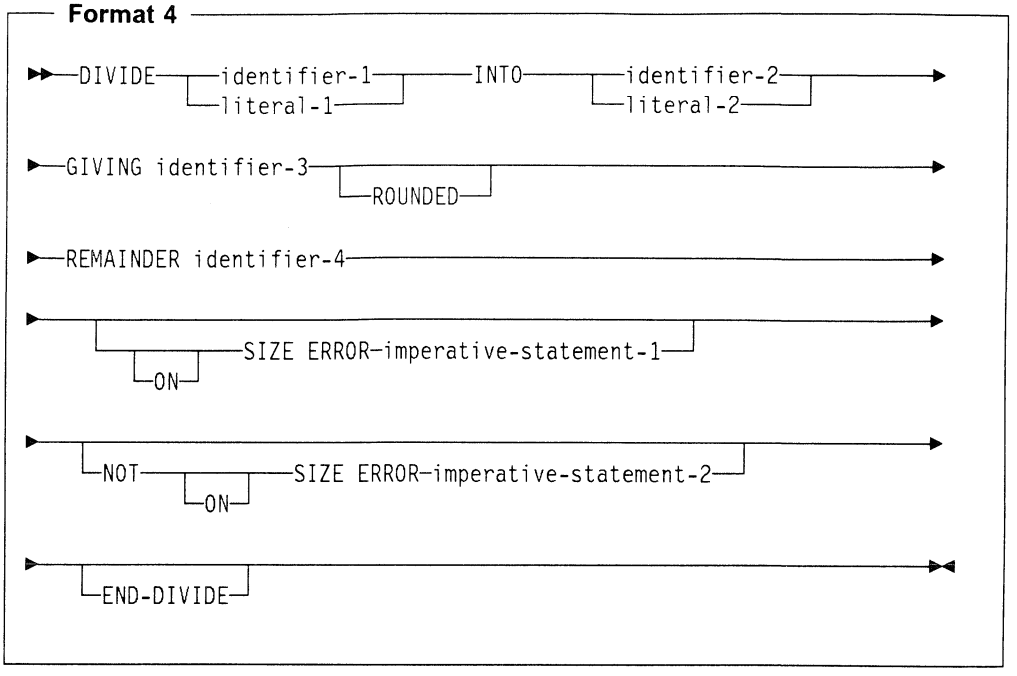


Format 3

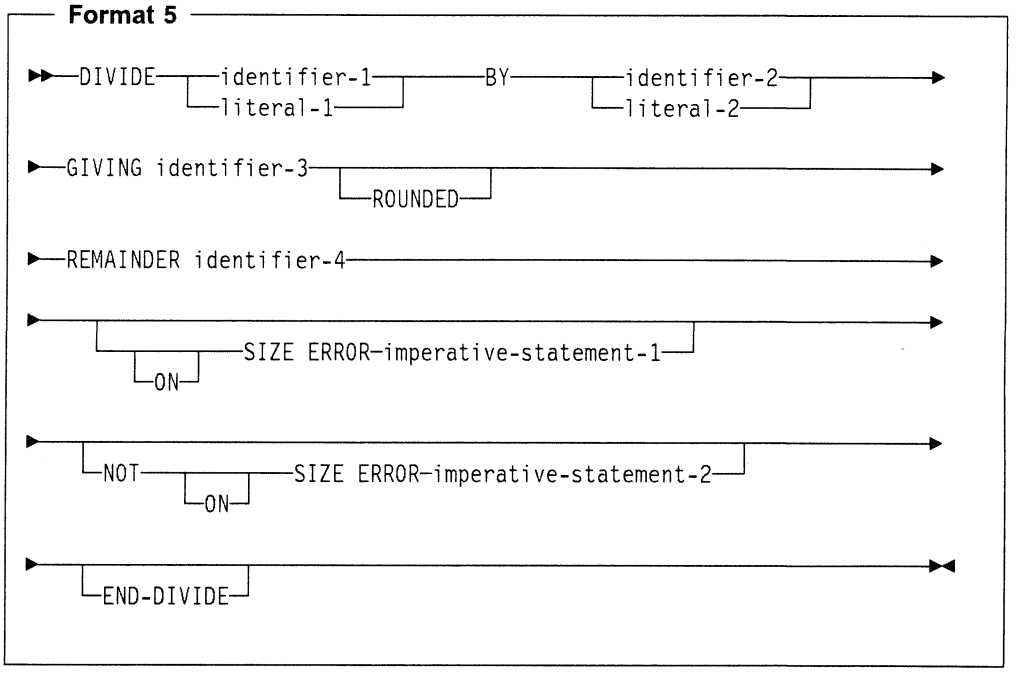


In Formats 2 and 3, the value of identifier-1 or literal-1 is divided into (Format 2) or by (Format 3) the value of identifier-2 or literal-2. The value of the quotient is stored in each data item referenced by identifier-3.

Format 4



Format 5



In Formats 4 and 5, the value of identifier-1 or literal-1 is divided into (Format 4) or by (Format 5) identifier-2 or literal-2. The value of the quotient is stored in identifier-3, and the value of the remainder is stored in identifier-4.

DIVIDE INTO Example: When IDENT-1 equals 2, QUOTIENT equals 5 after the processing of the following statement:

```
DIVIDE IDENT-1 INTO 10 GIVING QUOTIENT.
```

DIVIDE BY Example: When IDENT-1 equals 60, QUOTIENT equals 4 after the processing of the following statement:

```
DIVIDE IDENT-1 BY 15 GIVING QUOTIENT.
```

For all Formats:

identifier-1, identifier-2
Must name an elementary numeric item.

identifier-3, identifier-4
Must name an elementary numeric or numeric-edited item.

literal
Must be a numeric literal.

ROUNDED Phrase

For Formats 1, 2, and 3, see "ROUNDED Phrase" on page 210.

For Formats 4 and 5, the quotient used to calculate the remainder is in an intermediate field. The value of the intermediate field is truncated rather than rounded.

REMAINDER Phrase

The result of subtracting the product of the quotient and the divisor from the dividend is stored in identifier-4. If identifier-3, the quotient, is a numeric-edited item, the quotient used to calculate the remainder is an intermediate field that contains the unedited quotient.

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.

SIZE ERROR Phrases

For Formats 1, 2, and 3, see "SIZE ERROR Phrases" on page 210.

For Formats 4 and 5, if the size error occurs in the quotient, no remainder calculation is meaningful. Therefore, the contents of the quotient field (identifier-3) and the remainder field (identifier-4) are unchanged.

If size error occurs in the remainder, the contents of the remainder field (identifier-4) are unchanged.

In either of these cases, you must analyze the results to determine which situation has actually occurred.

For information on the NOT ON SIZE ERROR phrase, see page 212.

END-DIVIDE Phrase

This explicit scope terminator serves to delimit the scope of the DIVIDE statement. END-DIVIDE permits a conditional DIVIDE statement to be nested in another conditional statement. END-DIVIDE may also be used with an imperative DIVIDE statement.

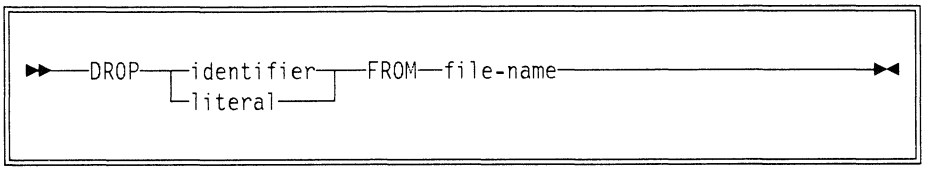
For more information, see "Delimited Scope Statements" on page 207.

IBM Extension

DROP Statement

The DROP statement releases a program device that has been acquired by a TRANSACTION file.

DROP Statement



literal
identifier
Literal or the contents of identifier indicates the program device name of the device to be dropped. Literal, if specified, must be nonnumeric and 10 characters or less in length. Identifier, if specified, must refer to an alphanumeric data item, 10 characters or less in length.

file-name
File-name must refer to a file with an organization of TRANSACTION, and the file must be open in order to be used in the DROP statement. If no DROP statement is issued, program devices attached to a TRANSACTION file are implicitly released when that file is finally closed.

Program devices specified in a DROP statement must have been acquired by the TRANSACTION file, either through an explicit ACQUIRE or through an implicit ACQUIRE at OPEN time.

After successful execution of the DROP statement, the program device is no longer available for input or output operations through the TRANSACTION file. The device may be reacquired if necessary. The contents of the record area associated with a released program device are no longer available, even if the device is reacquired.

If the DROP statement is unsuccessful, any applicable USE AFTER EXCEPTION/ERROR procedures are executed.

The DROP statement can also be used as an aid in recovering from I-O errors. For more information, see the Transaction File Recovery procedures in the *COBOL/400 User's Guide*.

End of IBM Extension

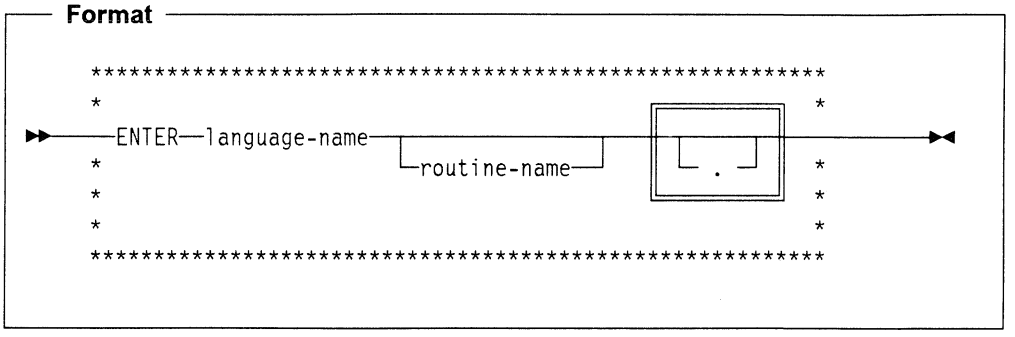
ENTER Statement

The ENTER statement allows the use of more than one source language in the same source program.

Note: The ENTER statement is an obsolete statement and is to be deleted from the next revision of the ANSI Standard. It is syntax-checked during compilation but has no effect on the execution of the program.

Note: The COBOL/400 compiler does not allow another programming language to be used in the source programs.

Format



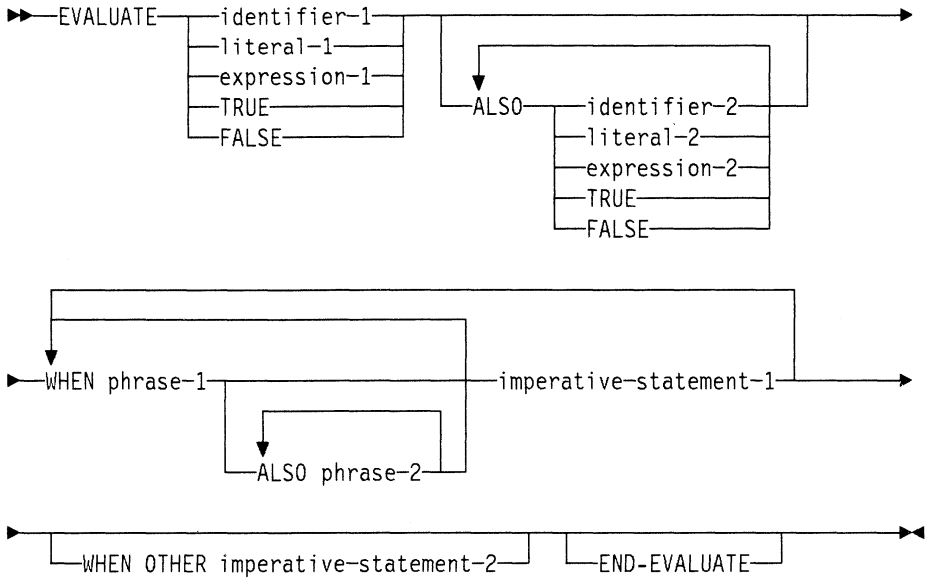
language-name
A system name that has no defined meaning. It must follow the rules for formation of a user-defined word. At least one character must be alphabetic.

routine-name
Must follow the rules for formation of a user-defined word. At least one character must be alphabetic.

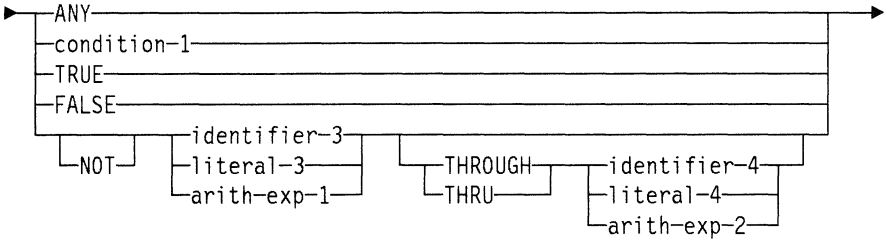
EVALUATE Statement

The EVALUATE statement provides a shorthand notation for a series of nested IF statements. It can evaluate multiple conditions. That is, the IF statements can be made up of compound conditions. The subsequent action of the object program depends on the results of these evaluations.

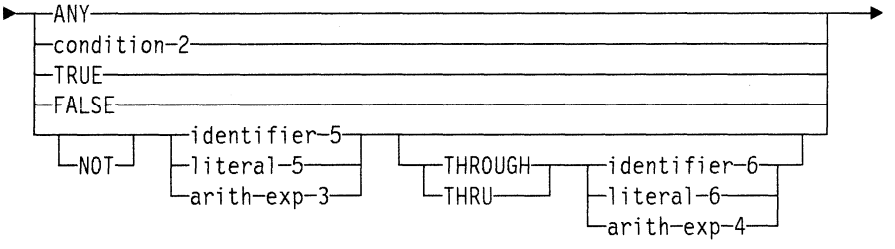
Format



where phrase-1 **is:**



where phrase-2 **is:**



The following example shows the coding for an EVALUATE statement and the equivalent coding for an IF statement.

Example of the EVALUATE Statement:

```
EVALUATE  MENU-INPUT
    WHEN "0"
        PERFORM INIT-PROC
    WHEN "1" THRU "9"
        PERFORM PROCESS-PROC
    WHEN "R"
        PERFORM READ-PARMS
    WHEN "X"
        PERFORM CLEANUP-PROC
    WHEN OTHER
        PERFORM ERROR-PROC
END-EVALUATE.
```

The Equivalent IF Statement:

```
IF (MENU-INPUT = "0") THEN
   PERFORM INIT-PROC
ELSE
   IF (MENU-INPUT ≥ "1") AND (MENU-INPUT ≤ "9") THEN
      PERFORM PROCESS-PROC
   ELSE
      IF (MENU-INPUT = "R") THEN
         PERFORM READ-PARMS
      ELSE
         IF (MENU-INPUT = "X") THEN
            PERFORM CLEANUP-PROC
         ELSE
            PERFORM ERROR-PROC
         END-IF
      END-IF
   END-IF
END-IF.
```

Following is a more complex example of an EVALUATE statement and the equivalent IF statement.

EVALUATE Statement - Complex Example:

```
EVALUATE  A = B  ALSO  C > D  ALSO  TRUE
    WHEN  TRUE   ALSO  TRUE   ALSO  E = F + 15
        imp-stat-1
    WHEN  TRUE   ALSO  TRUE   ALSO  E > 12
        imp-stat-2
    WHEN  TRUE   ALSO  FALSE  ALSO  ANY
        imp-stat-3
    WHEN  FALSE  ALSO  TRUE   ALSO  ANY
        imp-stat-4
    WHEN  FALSE  ALSO  FALSE  ALSO  ANY
        imp-stat-5
END-EVALUATE.
```

The Equivalent IF Statement:

```
IF A = B THEN
    IF C > D THEN
        IF E = F + 15 THEN
            imp-stat-1
        ELSE
            IF E > 12 THEN
                imp-stat-2
            END-IF
        END-IF
    ELSE
        imp-stat-3
    END-IF
ELSE
    IF C > D THEN
        imp-stat-4
    ELSE
        imp-stat-5
    END-IF
END-IF.
```

Operands before the WHEN phrase

Are interpreted in one of two ways, depending on how they are specified:

- Individually, they are called selection **subjects**.
- Collectively, they are called a **set** of selection subjects.

Operands in the WHEN phrase

Are interpreted in one of two ways, depending on how they are specified:

- Individually, they are called selection **objects**.
- Collectively, they are called a **set** of selection objects.

ALSO

Separates selection subjects within a set of selection subjects; separates selection objects within a set of selection objects.

THROUGH and THRU
Are equivalent.

Two operands connected by a THRU phrase must be of the same class. The two operands thus connected constitute a single selection object.

The number of selection objects within each set of selection objects must be equal to the number of selection subjects.

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:

- Identifiers, literals, or arithmetic expressions appearing within a selection object must be valid operands for comparison to the corresponding operand in the set of selection subjects.
- Condition-1, condition-2, or the word TRUE or FALSE appearing as a selection object must correspond to a conditional expression or the word TRUE or FALSE in the set of selection subjects.
- Condition-1, and condition-2 may be any form of a conditional expression.
- The word ANY may correspond to a selection subject of any type.

IBM Extension

- Where identifiers are permitted, they can reference items whose usage is implicitly or explicitly defined as POINTER.

End of IBM Extension

Conditional expressions may be simple or complex conditions.

END-EVALUATE Phrase

This explicit scope terminator serves to delimit the scope of the EVALUATE statement. END-EVALUATE permits a conditional EVALUATE statement to be nested in another conditional statement.

For more information, see "Delimited Scope Statements" on page 207.

Determining Values

The execution of the EVALUATE statement operates as if each selection subject and selection object were evaluated and assigned a numeric or nonnumeric value, a range of numeric or nonnumeric values, or a truth value. These values are determined as follows:

- Any selection subject specified by identifier-1, identifier-2, . . . and any selection object specified by identifier-3 and/or identifier-5 without the NOT or THRU phrase, are assigned the value and class of the data item that they reference.
- Any selection subject specified by literal-1, literal-2, . . . and any selection object specified by literal-3 and/or literal-5 without the NOT or THRU phrase, are assigned the value and class of the specified literal. If literal-3 and/or literal-5 is the figurative constant ZERO, it is assigned the class of the corresponding selection subject.

- Any selection subject in which expression-1, expression-2, . . . is specified as an **arithmetic** expression, and any selection object without the NOT or THRU phrase in which arithmetic-expression-1 and/or arithmetic-expression-3 is specified, are assigned numeric values according to the rules for evaluating an arithmetic expression. (See "Arithmetic Expressions" on page 187.)
- Any selection subject in which expression-1, expression-2, . . . is specified as a **conditional** expression, and any selection object in which condition-1 and/or condition-2 is specified, are assigned a truth value according to the rules for evaluating conditional expressions. (See "Conditional Expressions" on page 189.)
- 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.
- Any selection object specified by the word ANY is not further evaluated.
- If the THRU phrase is specified for a selection object without the NOT phrase, the range of values is all values that, when compared to the selection subject, are greater than or equal to the first operand and less than or equal to the second operand, according to the rules for comparison. If the first operand is greater than the second operand, there are no values in the range.
- If the NOT phrase is specified for a selection object, the values assigned to that item are all values not equal to the value, or range of values, that would have been assigned to the item had the NOT phrase been omitted.

Comparing Selection Subjects and Objects

The execution of the EVALUATE statement then proceeds as if the values assigned to the selection subjects and selection objects were compared to determine whether any WHEN phrase satisfies the set of selection subjects. This comparison proceeds as follows:

1. 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:
   a. If the items being compared are assigned numeric or nonnumeric values, or a range of numeric or nonnumeric values, the comparison is satisfied if the value, or one value in 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.
   b. If the items being compared are assigned truth values, the comparison is satisfied if the items are assigned identical truth values.
   c. 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.
2. 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.

3. If the above comparison is not satisfied for every selection object within the set of selection objects being compared, that set of selection objects does not satisfy the set of selection subjects.
4. 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.

Executing the EVALUATE Statement

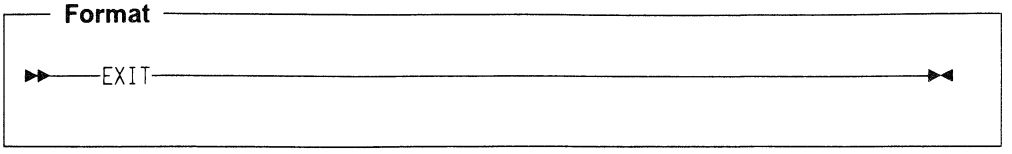
After the comparison operation is completed, execution of the EVALUATE statement proceeds as follows:

- If a WHEN phrase is selected, execution continues with the first imperative-statement-1 following the selected WHEN phrase. Note that multiple WHEN statements are allowed for a single imperative-statement-1.
- If no WHEN phrase is selected and a WHEN OTHER phrase is specified, execution continues with imperative-statement-2.
- If no WHEN phrase is selected and no WHEN OTHER phrase is specified, execution continues with the next executable statement following the scope delimiter.
- The scope of execution of the EVALUATE statement is terminated when execution reaches the end of the scope of the selected WHEN phrase or WHEN OTHER phrase, or when no WHEN phrase is selected and no WHEN OTHER phrase is specified.

EXIT Statement

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

Format



The EXIT statement assigns a name to a given point in a program. The EXIT statement has no other effect on the compilation or execution of the program. The EXIT statement must be preceded by a paragraph-name and must appear in a sentence by itself. This sentence must be the only sentence in the paragraph.

The EXIT statement is useful for documenting the end point in a series of paragraphs. If an EXIT paragraph is written as the last paragraph in a declarative procedure or a series of performed procedures, it identifies the point at which control will be transferred:

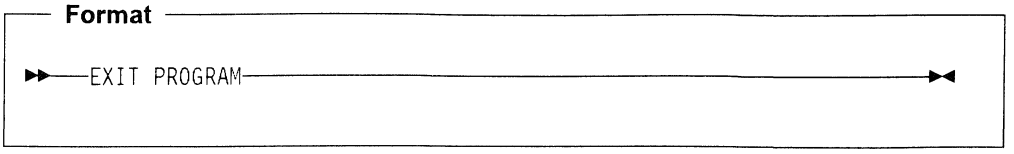
- When control reaches such an EXIT paragraph and the associated declarative or PERFORM statement is active, control is transferred to the appropriate part of the Procedure Division.
- When control reaches an EXIT paragraph that is not the end of a range of procedures governed by an active PERFORM or USE statement, control passes through the EXIT statement to the first statement of the next paragraph.

Without an EXIT statement, the end of the sequence is difficult to determine, unless you know the logic of the program.

EXIT PROGRAM Statement

The EXIT PROGRAM statement specifies the end of a called program and returns control to the calling program.

Format



If control reaches an EXIT PROGRAM statement while operating as a subprogram in a COBOL run unit, control returns to the point in the calling program immediately following the CALL statement. The program state of the calling program is identical to that which existed at the time it executed the CALL statement. 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.

If control reaches an EXIT PROGRAM statement, and no CALL statement is active, control passes through the exit point to the next executable statement.

The EXIT PROGRAM statement must appear as the only statement, or as the last statement, in a series of imperative statements in a sentence.

When there is no next executable statement in a called program, an implicit EXIT PROGRAM statement is executed.

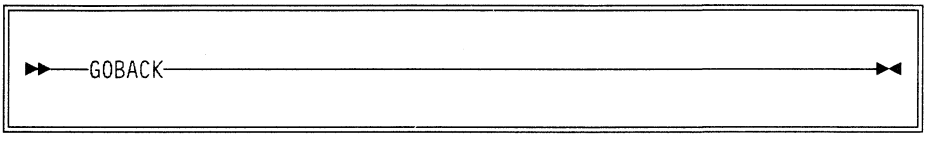
IBM Extension

GOBACK Statement

The GOBACK statement functions like the EXIT PROGRAM statement when it is coded as part of a program that is a subprogram in a COBOL run unit, and like the STOP RUN statement when coded in a program that is a main program in a COBOL run unit.

The GOBACK statement specifies the logical end of a program.

Format



A GOBACK statement should appear as the only statement, or as the last of a series of imperative statements, in a sentence, because any statements following the GOBACK are not executed.

If control reaches a GOBACK statement in a subprogram in a COBOL run unit, control returns to the point in the calling program immediately following the CALL statement, as in the EXIT PROGRAM statement.

The table below shows the action taken for the GOBACK statement in both a main program and a subprogram in a COBOL run unit.

Termination Statement	Main Program	Subprogram
GOBACK	Ends COBOL run unit.	Return to calling program.

For further information about COBOL run units, see the chapter on Interprogram Communication in the *COBOL/400 User's Guide*.

End of IBM Extension

GO TO Statement

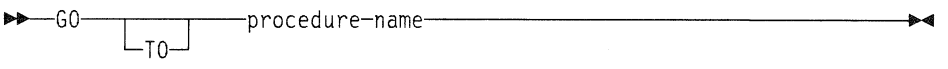
The GO TO statement transfers control from one part of the Procedure Division to another. There are three types of GO TO statements:

1. Unconditional
2. Conditional
3. Altered.

Unconditional GO TO

The unconditional GO TO statement transfers control to the first statement in the paragraph or section named in procedure-name, unless the GO TO statement has been modified by an ALTER statement. (See "ALTER Statement" on page 249.)

Format 1—Unconditional



procedure-name
Must name a procedure or a section in the Procedure Division.

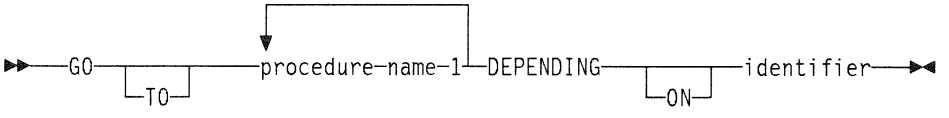
An unconditional GO TO statement, when it appears in a sequence of imperative statements, must be the last statement in the sequence.

When a paragraph is referred to by an ALTER statement, the paragraph must consist of a paragraph-name followed by an unconditional or altered GO TO statement.

Conditional GO TO

The conditional GO TO statement transfers control to one of a series of procedures, depending on the value of the identifier.

Format 2—Conditional



procedure-name-1
Must name a procedure or a section in the Procedure Division.

identifier
Must be a numeric elementary data item which is an integer.

If 1, control is transferred to the first statement in the procedure named by the first occurrence of procedure-name-1;

If 2, control is transferred to the first statement in the procedure named by the second occurrence of procedure-name-1, and so forth.

If the value of identifier is anything other than a value within the range of 1 through n (where n is the number of procedure-names specified in this GO TO statement), no control transfer occurs. Instead, control passes to the next statement in the normal sequence of execution.

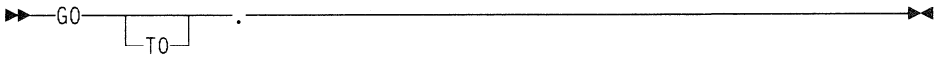
The maximum number of procedure-names permitted for a conditional GO TO statement is 255.

Altered GO TO

The altered GO TO statement transfers control to the first statement of the paragraph named in the ALTER statement. The altered GO TO statement is an obsolete element and is to be deleted from the next revision of the ANSI Standard.

An ALTER statement referring to the paragraph containing this GO TO statement must have been executed before this GO TO statement is executed.

Format 3—Altered

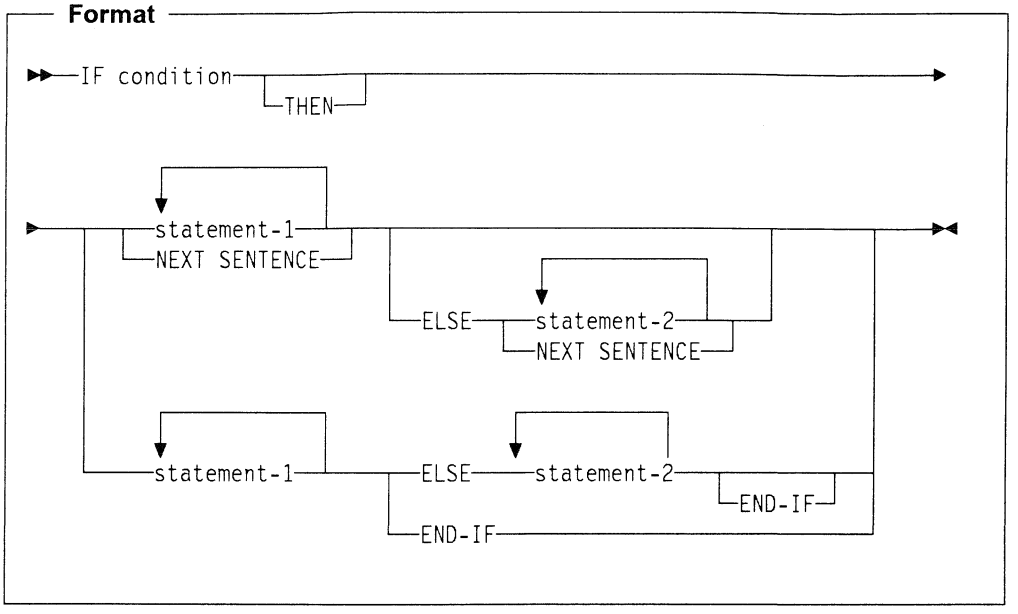


When an ALTER statement refers to a paragraph, the paragraph may consist only of the paragraph-name followed by an unconditional or altered GO TO statement.

IF Statement

The IF statement evaluates a condition and provides for alternative actions in the object program, depending on the evaluation.

Format



condition
May be any simple or complex condition, as described in "Conditional Expressions" on page 189.

statement-1, statement-2
Can be any one of the following:

- An imperative statement
- A conditional statement
- An imperative statement followed by a conditional statement.

NEXT SENTENCE
If the END-IF phrase is specified, the NEXT SENTENCE phrase must not be specified.

IBM Extension

NEXT SENTENCE can be specified with END-IF.

End of IBM Extension

ELSE NEXT SENTENCE
May be omitted if it immediately precedes a separator period that ends the IF statement.

END-IF Phrase

This explicit scope terminator serves to delimit the scope of the IF statement. END-IF permits a conditional IF statement to be nested in another conditional statement.

For more information, see "Delimited Scope Statements" on page 207.

Transferring Control

If condition tested is **true**, one of the following actions takes place:

- Statement-1, if specified, is executed. If statement-1 contains a procedure branching statement, control is transferred, according to the rules for that statement. If statement-1 does not contain a procedure-branching statement, the ELSE phrase, if specified, is ignored, and control passes to the next executable statement after the corresponding END-IF or separator period.
- NEXT SENTENCE, if specified, is executed; that is, the ELSE phrase, if specified, is ignored, and control passes to the statement following the closest separator period.

If the condition tested is **false**, one of the following actions takes place:

- ELSE statement-2, if specified, is executed. If statement-2 contains a procedure-branching statement, control is transferred, according to the rules for that statement. If statement-2 does not contain a procedure-branching statement, control is passed to the next executable statement after the corresponding END-IF or separator period.
- ELSE NEXT SENTENCE, if specified, is executed and control passes to the statement following the closest separator period.
- If ELSE NEXT SENTENCE is omitted, control passes to the next executable statement after the corresponding END-IF or separator period.

Note: When ELSE or ELSE NEXT SENTENCE are omitted, all statements following the condition and preceding the corresponding END-IF or the separator period for the sentence are considered to be part of statement-1.

Nested IF Statements

The presence of one or more IF statements within the initial IF statement constitutes a "nested IF statement." Nesting statements is much like specifying subordinate arithmetic expressions enclosed in parentheses and combined in larger arithmetic expressions.

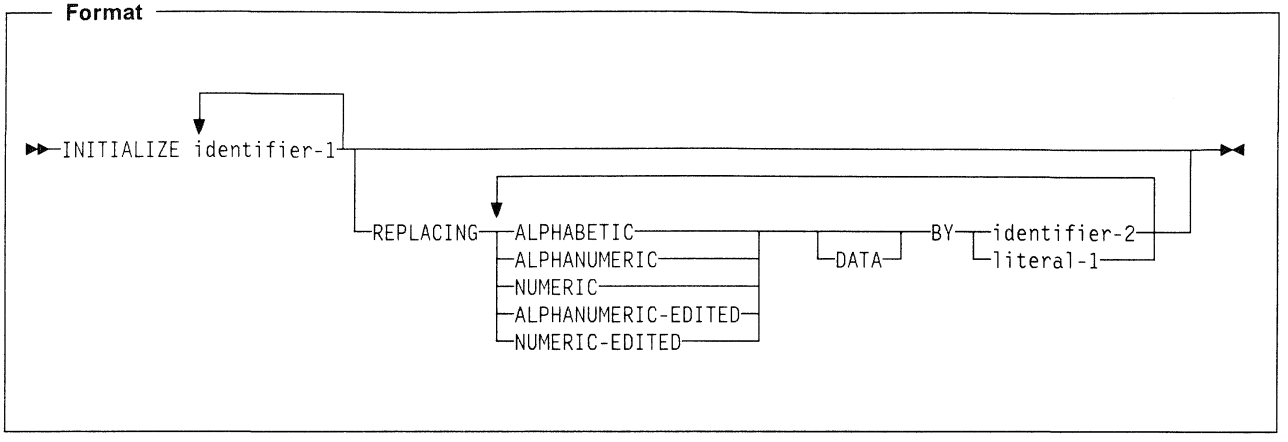
Statement-1 and statement-2 in IF statements may consist of one or more imperative statements and/or a conditional statement. If an IF statement appears as statement-1 or statement-2, or as part of statement-1 or statement-2, it is said to be **nested**.

IF statements contained within IF statements are 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.

INITIALIZE Statement

The INITIALIZE statement sets selected categories of data fields to predetermined values. It is functionally equivalent to one or more MOVE statements.

Format



identifier-1
Receiving area(s).

identifier-2, literal-1
Sending area(s).

A subscripted or reference-modified item can be specified for identifier-1. A complete table may be initialized only by specifying identifier-1 as a group that contains the complete table.

Neither identifier-1 nor any item subordinate to it may contain the DEPENDING ON phrase of the OCCURS clause. The data description entry for identifier-1 must not contain a RENAMES clause. An index data item may not be an operand of INITIALIZE.

REPLACING Phrase

When the REPLACING phrase is used:

- The category of identifier-2 or literal-1 must be compatible with the category indicated in the corresponding REPLACING phrase, according to the rules for MOVE.
- The same category cannot be repeated in a REPLACING phrase.
- The key word following the word REPLACING corresponds to a category of data shown in "Classes and Categories of Data" on page 106.

When the REPLACING phrase is not used:

- SPACE is the implied sending field for alphabetic, alphanumeric, and alphanumeric-edited items.
- ZERO is the implied sending field for numeric, and numeric-edited items.

INITIALIZE Statement Rules

1. Whether identifier-1 references an elementary or group item, all operations are performed as if a series of MOVE statements had been written, each of which had an elementary item as a receiving field.

   If the REPLACING phrase is specified:

   - 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 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 if 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, with the following exceptions:

   - Index and pointer data items
   - Elementary FILLER data items
   - Items that are subordinate to identifier-1 and contain a REDEFINES clause, or any items subordinate to such an item. (However, identifier-1 may contain a REDEFINES clause or be subordinate to a redefining item.)
   - BOOLEAN data items.

2. The areas referenced by identifier-1 are initialized in the order (left to right) of the appearance of identifier-1 in the statement. Within a group receiving field, affected elementary items are initialized in the order of their definition within the group.
3. If identifier-1 occupies the same storage area as identifier-2, the result of the execution of this statement is undefined, even if these operands are defined by the same data description entry.
4. If identifier-1 is a group item, then all of the items within that group item are considered as being referenced in the program.

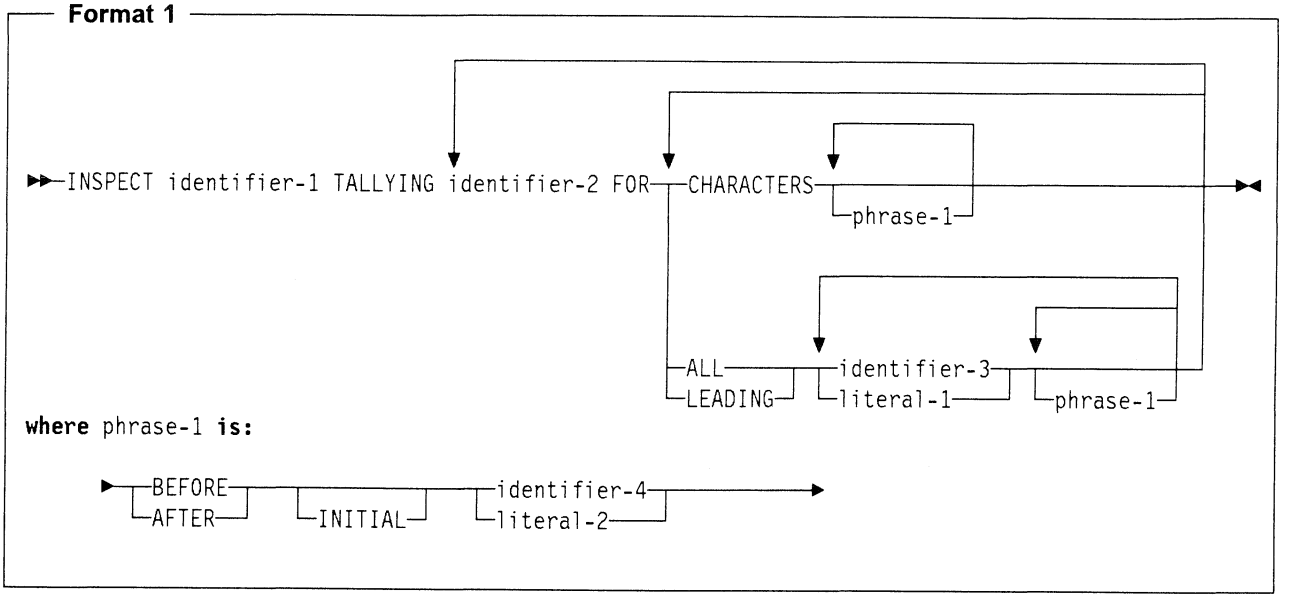
INSPECT Statement

The INSPECT statement specifies that characters in a data item are to be counted (tallied) or replaced or both:

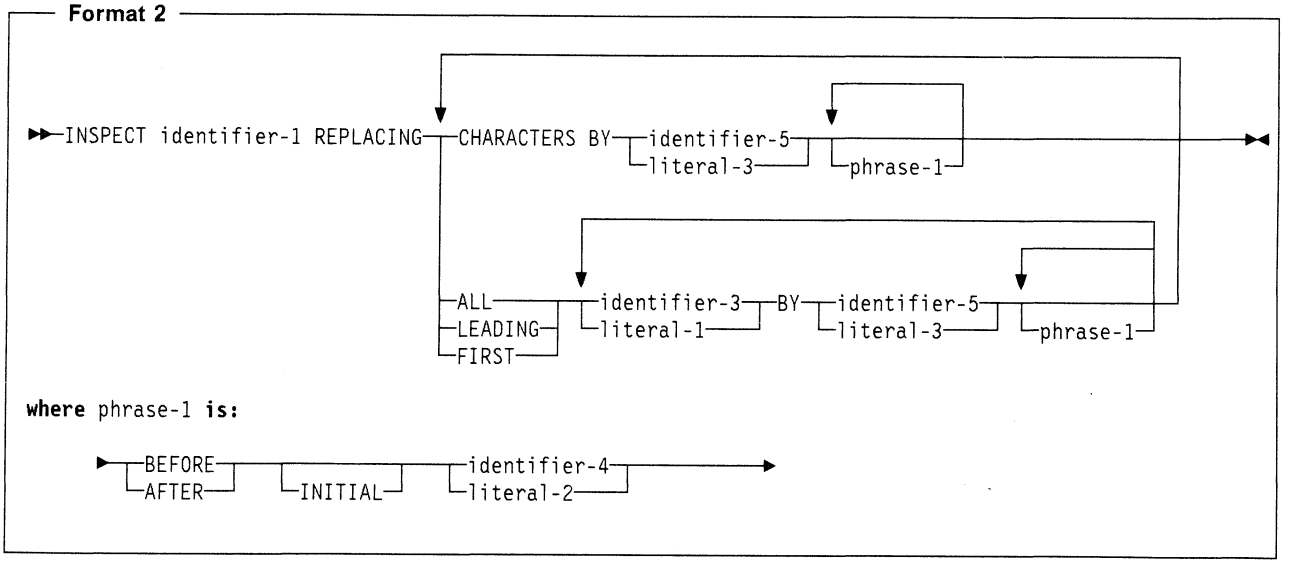
- It will count the occurrence of a specific character (alphabetic, numeric, or special character) in a data item.
- It will fill all or portions of a data item with spaces or zeros.
- It will translate characters from one collating sequence to another.

  This statement does not support reference modification.

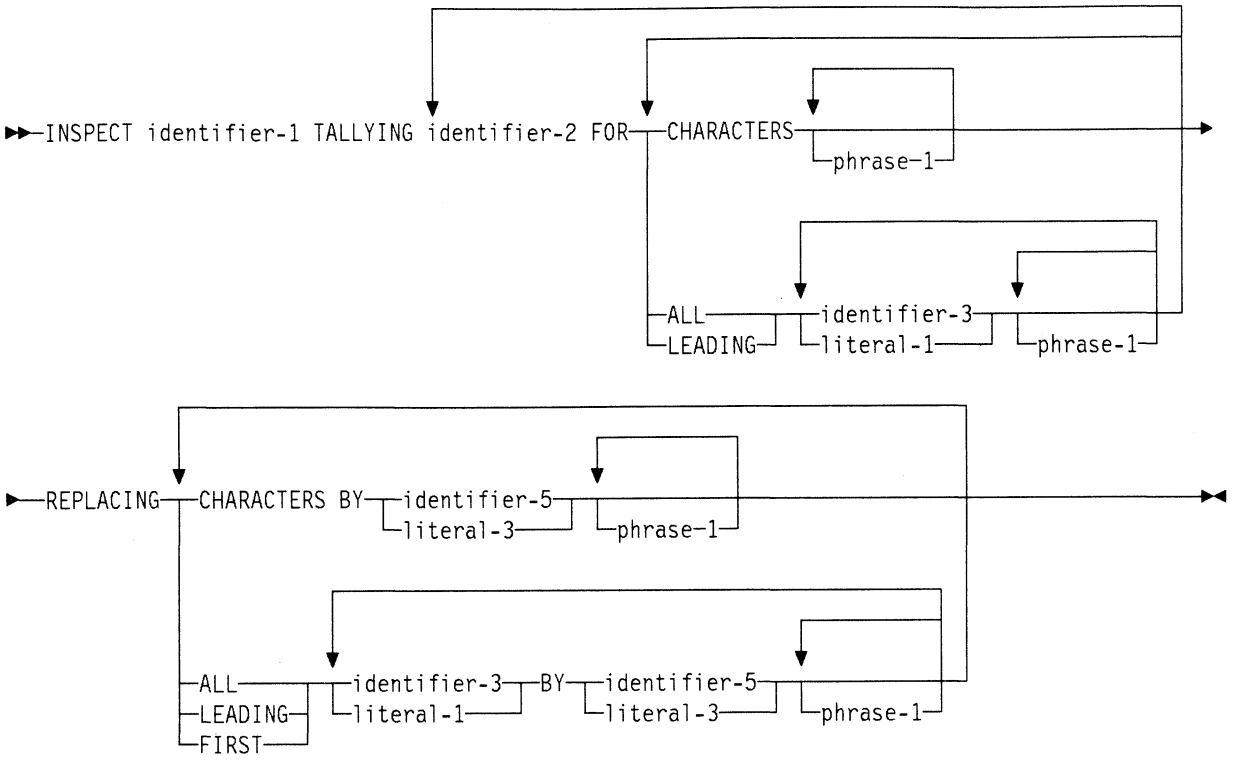
Format 1



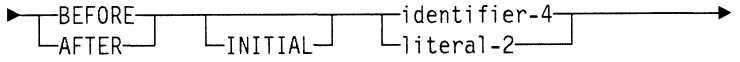
Format 2



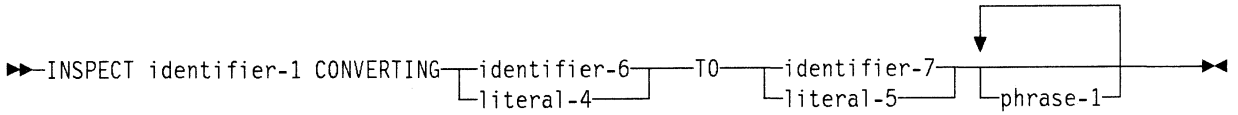
Format 3



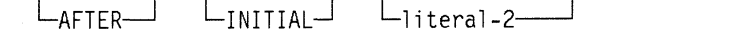
where phrase-1 **is:**



Format 4



where phrase-1 **is:**



identifier-1
The inspected item. It must be an elementary or group item with USAGE DISPLAY.

identifier-2
Must be an elementary numeric data item.

identifier-3 . . . identifier-7
Must be elementary data items with USAGE DISPLAY.

For use in the INSPECT statement, the content of each data item referenced by all identifiers except identifier-2 (the count field) will be treated as follows:

ALPHABETIC OR ALPHANUMERIC ITEM
Treated as a character string.

ALPHANUMERIC-EDITED, NUMERIC-EDITED, OR UNSIGNED NUMERIC (EXTERNAL DECIMAL) ITEM
Treated as if defined as alphanumeric with the INSPECT statement referring to the alphanumeric item.

SIGNED NUMERIC (EXTERNAL DECIMAL) ITEM
Treated as if moved to an unsigned external decimal item of the same length, and then redefined as alphanumeric, with the INSPECT statement referring to the alphanumeric item. If the sign is a separate character, the byte containing the sign is not examined and, therefore, not replaced.

literal-1 . . . literal-5
Must be nonnumeric and may be any figurative constant that does not begin 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.

Except when the BEFORE or AFTER phrase is specified, inspection begins at the leftmost character position of the inspected item (identifier-1) and proceeds character-by-character to the rightmost position.

The operands of the following phrases are compared in the left-to-right order in which they are specified in the INSPECT statement:

- TALLYING (literal-1 or identifier-3, . . .)
- REPLACING (literal-3 or identifier-5, . . .)

Subscripting associated with any identifier is evaluated only once as the first operation in the execution of the INSPECT statement.

Comparison Rules

In the following list of comparison rules, the TALLYING and REPLACING phrases are the items being compared:

1. When both the TALLYING and REPLACING phrases are specified, the INSPECT statement is executed as if an INSPECT TALLYING statement were specified, immediately followed by an INSPECT REPLACING statement.
2. The first comparand is compared with an equal number of leftmost contiguous characters in the inspected item. The comparand matches the inspected characters only if both are equal, character-for-character.
3. If no match occurs for the first comparand, the comparison is repeated for each successive comparand until either a match is found or all comparands have been acted upon.

4. If a match is found, tallying or replacing takes place, as described in the following TALLYING/REPLACING phrase descriptions. In the inspected item, the first character following the rightmost matching character is now considered to be in the leftmost character position. The process described in rules 2 and 3 is then repeated.
5. If no match is found, then, in the inspected item, the first character following the leftmost inspected character is now considered to be in the leftmost character position. The process described in rules 2 and 3 is then repeated.
6. If the CHARACTERS phrase is specified, an implied one-character item is used in the process described in rules 2 and 3. The implied character is always considered to match the inspected character in the inspected item.
7. The actions taken in rules 1 through 6 (defined as the **comparison cycle**) are repeated until the rightmost character in the inspected item has either been matched or has been considered as being in the leftmost character position. Inspection is then terminated.

When the BEFORE or AFTER phrase is specified, the preceding rules are modified, as described in "BEFORE and AFTER Phrases (All Formats)" on page 313.

Figure 17 on page 310 is an example of INSPECT statement results.

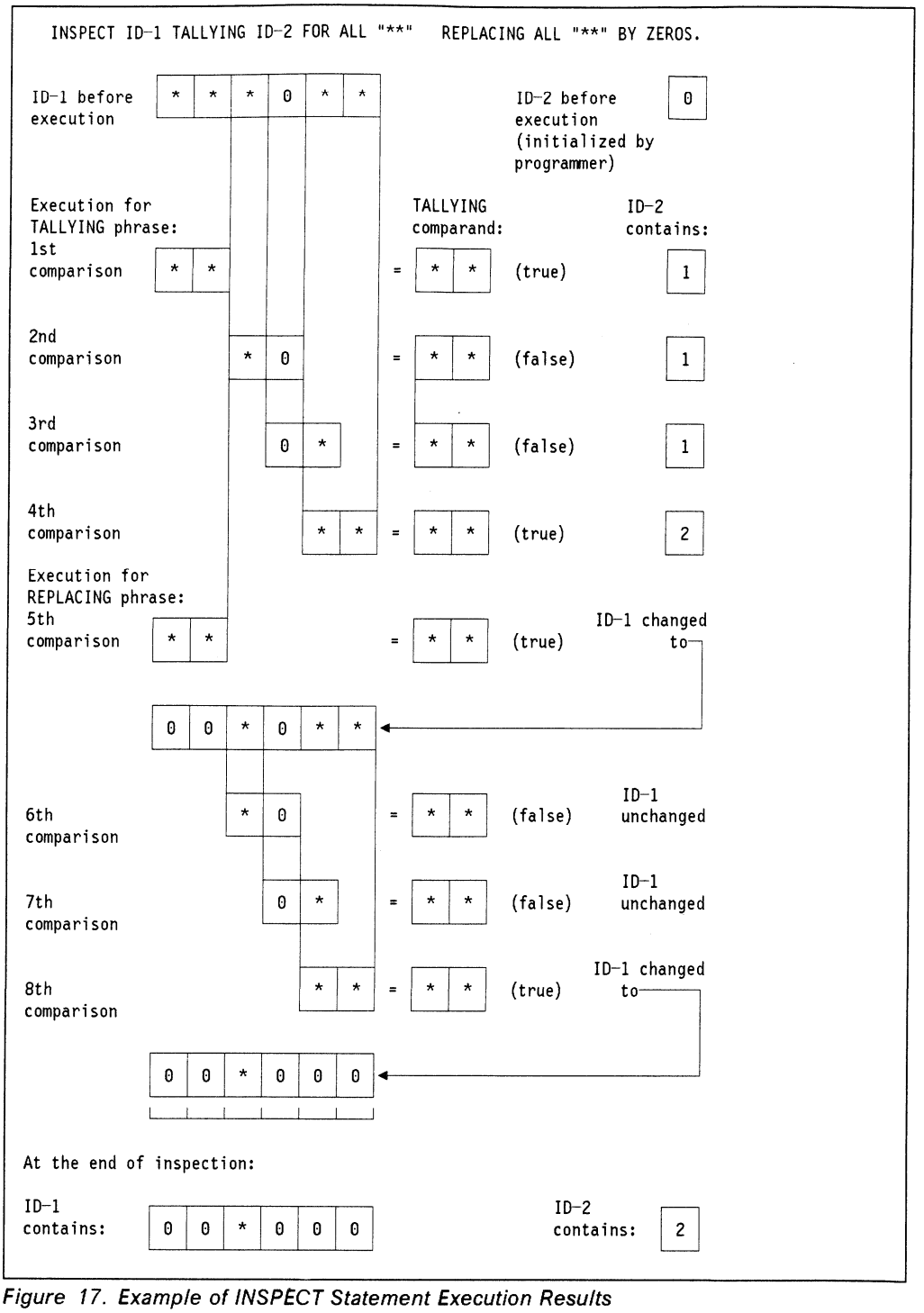


Figure 17. Example of INSPECT Statement Execution Results

INSPECT Example

The following example shows an INSPECT statement.

```
.. 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7

  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  ID-1      PIC X(10)   VALUE "ACADEMIANS".
  01  CONTR-1    PIC 99     VALUE 00.
  01  CONTR-2    PIC 99     VALUE ZEROS.
  PROCEDURE DIVISION.
 *    THIS ILLUSTRATES AN INSPECT STATEMENT WITH 2 VARIABLES.
  100-BEGIN-PROCESSING.
      DISPLAY CONTR-1 SPACE CONTR-2.
  101-MAINLINE-PROCESSING.
      PERFORM COUNT-IT THRU COUNT-EXIT.
      STOP RUN.
  COUNT-IT.
      INSPECT ID-1
      TALLYING CONTR-1
       FOR CHARACTERS BEFORE INITIAL "AD"
              CONTR-2
       FOR ALL "MIANS".
  DISPLAY-COUNTS.
      DISPLAY "CONTR-1 = " CONTR-1.
      DISPLAY "CONTR-2 = " CONTR-2.
      DISPLAY "*********EOJ*********".
  COUNT-EXIT.
      EXIT.
```

Resultant Output:

```
      00 00
      CONTR-1 = 02
      CONTR-2 = 01
      *********EOJ*********
```

TALLYING Phrase (Formats 1 and 3)

identifier-2
The count field. It must be an elementary integer item defined without the symbol P in its PICTURE character-string. You must initialize identifier-2 before the INSPECT statement is executed.

identifier-3 or literal-1
The tallying operand. If the tallying operand is a figurative constant, it is considered to be a 1-character nonnumeric literal.

When neither the BEFORE nor AFTER phrase is specified, the following actions take place when the INSPECT TALLYING statement is executed:

- If ALL is specified, the count field is increased by 1 for each nonoverlapping occurrence in the inspected item of this tallying operand, beginning at the leftmost character position and continuing to the rightmost.
- If LEADING is specified, the count field is increased by 1 for each contiguous nonoverlapping occurrence of this tallying operand in the inspected item, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which this tallying operand is eligible to participate.
- If CHARACTERS is specified, the count field is increased by 1 for each character (including the space character) in the inspected item. Thus, execution of the INSPECT TALLYING statement increases the value in the count field by the number of characters in the inspected item.

REPLACING Phrase (Formats 2 and 3)

identifier-3 or literal-1
The subject field.

identifier-5 or literal-3
The substitution field.

The subject field and the substitution field must be the same length. The following replacement rules apply:

- If the subject field is a figurative constant, it is considered to be a 1-character nonnumeric literal. Each character in the inspected item equivalent to the figurative constant is replaced by the single-character substitution field, which must be 1 character in length.
- If the substitution field is a figurative constant, the substitution field is considered to be the same length as the subject field. Each nonoverlapping occurrence of the subject field in the inspected item is replaced by the substitution field.
- When the subject and substitution fields are character-strings, each nonoverlapping occurrence of the subject field in the inspected item is replaced by the character-string specified in the substitution field.
- Once replacement has occurred in a given character position in the inspected item, no further replacement for that character position is made in this execution of the INSPECT statement.

When the CHARACTERS phrase is used, literal-3 or identifier-5 must be 1 character in length.

When neither the BEFORE nor AFTER phrase is specified, the following actions take place when the INSPECT REPLACING statement is executed:

- If CHARACTERS is specified, the substitution field must be 1 character in length. Each character in the inspected field is replaced by the substitution field, beginning at the leftmost character and continuing to the rightmost.
- If ALL is specified, each nonoverlapping occurrence of the subject field in the inspected item is replaced by the substitution field, beginning at the leftmost character and continuing to the rightmost.
- If LEADING is specified, each contiguous nonoverlapping occurrence of the subject field in the inspected item is replaced by the substitution field, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which this substitution field is eligible to participate.
- If FIRST is specified, the leftmost occurrence of the subject field in the inspected item is replaced by the substitution field.

BEFORE and AFTER Phrases (All Formats)

No more than one BEFORE phrase and one AFTER phrase can be specified for any one ALL, LEADING, CHARACTERS, FIRST or CONVERTING phrase. When these phrases are specified, the preceding rules for counting and replacing are modified.

identifier-4, literal-2

These are not counted or replaced. However, counting and/or replacing of the inspected item is bounded by the presence of the identifiers and literals. If the delimiter (identifier-4 or literal-2) is a figurative constant, it is considered to be 1 character in length.

When BEFORE is specified, counting and/or replacing of the inspected item begins at the leftmost character and continues until the first occurrence of the delimiter is encountered. If no delimiter is present in the inspected item, counting and/or replacing continues toward the rightmost character.

When AFTER is specified, counting and/or replacing of the inspected item begins with the first character to the right of the delimiter and continues toward the rightmost character in the inspected item. If no delimiter is present in the inspected item, no counting or replacement takes place.

CONVERTING Phrase (Format 4)

A string of replacement values may be expressed by the phrase. The size of the receiving location (identifier-7 or literal-5) must be the same size as the sending location (identifier-6 or literal-4). When a figurative constant is used as literal-5, the size of the figurative constant is equal to the size of literal-4 or identifier-6. The same character must not appear more than once either in literal-4 or identifier-6.

A Format 4 INSPECT statement is interpreted and executed as if a Format 2 INSPECT statement had been written with a series of ALL phrases (one for each character of literal-4), specifying the same identifier-1. The effect is as if each single character of literal-4 were referenced as literal-1, and the corresponding single character of literal-5 referenced as literal-3. Correspondence between the characters of literal-4 and the characters of literal-5 is by ordinal position within the data item.

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.

INSPECT Statement Examples

The following examples illustrate some uses of the INSPECT statement. In all instances, the programmer has initialized the COUNTR field to zero before the INSPECT statement is executed.

INSPECT ID-1 REPLACING CHARACTERS BY ZERO.

ID-1 Before	COUNTR After	ID-1 After
1234567	0	0000000
HIJKLMN	0	0000000

INSPECT ID-1 TALLYING COUNTR FOR CHARACTERS REPLACING CHARACTERS BY SPACES.

ID-1 Before	COUNTR After	ID-1 After
1234567	7	
HIJKLMN	7	

INSPECT ID-1 REPLACING CHARACTERS BY ZEROS BEFORE INITIAL QUOTE.

ID-1 Before	COUNTR After	ID-1 After
456"ABEL	0	000"ABEL
ANDES"12	0	00000"12
"TWAS BR	0	"TWAS BR

INSPECT ID-1 TALLYING COUNTR FOR CHARACTERS AFTER INITIAL "S" REPLACING ALL "A" BY "O".

ID-1 Before	COUNTR After	ID-1 After
ANSELM	3	ONSELM
SACKET	5	SOCKET
PASSED	3	POSSED

INSPECT ID-1 TALLYING COUNTR FOR LEADING "0" REPLACING FIRST "A" BY "2" AFTER INITIAL "C".

ID-1 Before	COUNTR After	ID-1 After
00ACADEMY00	2	00AC2DEMY00
0000ALABAMA	4	0000ALABAMA
CHATAM0000	0	CH2THAM0000

INSPECT ID-1 CONVERTING "ABCD" TO "XYZX" AFTER QUOTE BEFORE "#".

ID-1 Before	ID-1 After
AC"AEBDFBCD#AB"D	AC"XEYXFYZX#AB"D

MERGE Statement

The MERGE statement combines two or more identically sequenced files (that is, files that have already been sorted according to an identical set of ascending/descending keys) on one or more keys and makes records available in merged order to an output procedure or output file.

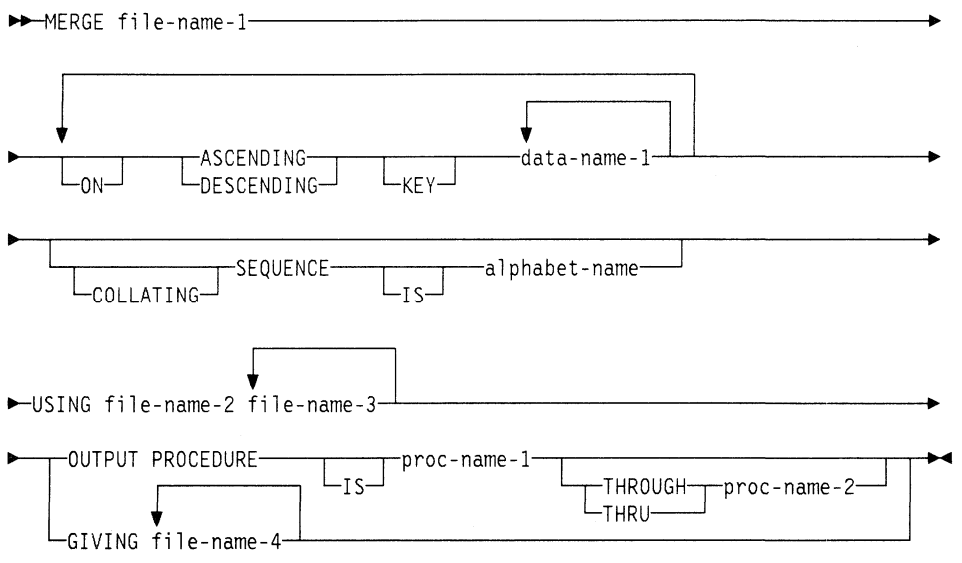
A MERGE statement may appear anywhere in the Procedure Division except in a Declarative Section. The maximum number of USING or GIVING files is 31.

IBM Extension

It is not necessary to sequence input files prior to a merge operation.

End of IBM Extension

Format



file-name-1

The name given in the SD entry that describes the record.

No file-name may be repeated in the MERGE statement.

When the MERGE statement is executed, all records contained in file-name-2, file-name-3, . . . are accepted by the merge program and then merged according to the key(s) specified.

Null-capable fields are supported, but null values are not. Null values result in a file status of 90.

ASCENDING/DESCENDING KEY Phrase

This phrase specifies that records are to be processed in an ascending or descending sequence (depending on the phrase specified), based on the specified merge keys.

KEY

KEY data items are listed in order of decreasing significance, no matter how they are divided into KEY phrases. Using the format above as an example:

data-name-1

Is a key data-name. Records are processed in ascending or descending order on this key.

Data-name-1 specifies a KEY data item on which the merge will be based. Each such data-name must identify a data item in a record associated with file-name-1. 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 left-most data-name is the major key, the next data-name is the next most significant key, and so forth.

The following rules apply:

- A specific KEY data item must be physically located in the same position and have the same data format in each input file; however, it need not have the same data-name.
- If file-name-1 has more than one record description, the KEY data items need be described in only one of the record descriptions.
- KEY data items must be fixed-length items.
- KEY data items must not contain an OCCURS clause or be subordinate to an item that contains an OCCURS clause.
- KEY data items can be qualified or reference modified; they cannot be subscripted or indexed.
- The total length (in bytes) of the KEY data items must not exceed 256.

The direction of the merge operation depends on the specification of the ASCENDING or DESCENDING key words as follows:

- When ASCENDING is specified, the sequence is from the lowest key value to the highest key value.
- When DESCENDING is specified, the sequence is from the highest key value to the lowest.
- If the KEY data item is alphabetic, alphanumeric, alphanumeric-edited, or numeric-edited, the sequence of key values depends on the collating sequence used (see "COLLATING SEQUENCE Phrase" on page 318 below).

The key comparisons are performed according to the rules for comparison of operands in a relation condition (see "Relation Condition" on page 192).

COLLATING SEQUENCE Phrase

This phrase specifies the collating sequence to be used in nonnumeric comparisons for the KEY data items in this merge operation.

alphabet-name

Must be specified in the SPECIAL-NAMES paragraph alphabet-name clause. Any one of the alphabet-name clause phrases can be specified with the following results:

- When NATIVE is specified, the EBCDIC collating sequence is used for all nonnumeric comparisons.
- When the literal phrase is specified, the collating sequence established by the specification of literals in the alphabet-name clause is used for all nonnumeric comparisons.
- When STANDARD-1 is specified, the ASCII collating sequence is used for all nonnumeric comparisons.
- When STANDARD-2 is specified, the International Reference Version of the ISO 7-bit code defined in International Standard 646, 7-bit Coded Character Set for Information Processing Interchange is used.

When the COLLATING SEQUENCE phrase is omitted, the PROGRAM COLLATING SEQUENCE clause (if specified) in the OBJECT-COMPUTER paragraph specifies the collating sequence to be used.

When both the COLLATING SEQUENCE phrase and the PROGRAM COLLATING SEQUENCE clause are omitted, the EBCDIC collating sequence is used.

USING Phrase

When the USING phrase is specified, all the records on file-name-2, file-name-3,... (that is, the input files) are transferred automatically to file-name-1. At the time the MERGE statement is executed, these files must not be open; the compiler generates code that opens, reads and closes the input files automatically. If EXCEPTION/ERROR procedures are specified for these files, the COBOL compiler makes the necessary linkage to these procedures.

All input files must be described in an FD entry in the Data Division, and their record descriptions must describe records of the same size as the record described for the merge file. If the elementary items that make up these records are not identical, input records must have an equal number of character positions as the merge record.

The input files must have sequential, relative or indexed organization.

GIVING Phrase

When the GIVING phrase is specified, all the merged records in file-name-1 are automatically transferred to the output file (file-name-4). At the start of execution of the MERGE statement, the file referenced by file-name-4 must not be open. For each of the files referenced by file-name-4, the execution of the MERGE statement causes the following actions to be taken:

- The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase had been executed.
- 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. The records overwrite the previous contents, if any, of the file.

IBM Extension

If file-name-1 is a logical database file, the records are added to the end of the file.

End of IBM Extension

If the file referenced by file-name-4 is an INDEXED file then the associated key data-name for that file must have an ASCENDING KEY phrase in the merge statement. This same data-name must occupy the identical character positions in its record as the data item associated with the prime record key for the file.

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', and so on. After execution of the MERGE statement, the content of the relative key data item indicates the last record returned to the file.

- The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

Note: When duplicate keys are found when writing to an indexed file, the MERGE will terminate and the merged data in all GIVING files will be incomplete.

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-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.

The output file must be described in an FD entry in the Data Division, and its record description(s) must describe records of the same size as the record described for the merge file. If the elementary items that make up these records are not identical, the output record must have an equal number of character positions as the merge record.

The output file must have a sequential, relative or indexed organization.

The output file should also be created without a keyed sequence access path. When the output file has such a path, the MERGE statement cannot override the collating sequence defined in the data description specifications (DDS).

OUTPUT PROCEDURE Phrase

This phrase specifies the procedure-name(s) of a procedure that is to select or modify output records from the merge operation.

proc-name-1
Specifies the first (or only) section in the OUTPUT PROCEDURE.

proc-name-2
Identifies the last section of the OUTPUT PROCEDURE.

Procedural statements within the OUTPUT PROCEDURE are restricted as follows:

- The OUTPUT PROCEDURE must not contain any SORT or MERGE statements.

The OUTPUT PROCEDURE must consist of one or more sections that are written consecutively and do not form a part of any INPUT PROCEDURE. The OUTPUT PROCEDURE must include at least one RETURN statement in order to make merged records available for processing.

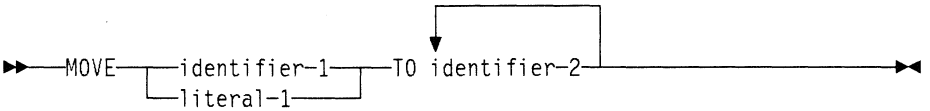
When an OUTPUT PROCEDURE is specified, control passes to it after the merge file (file-name-1) has been placed in sequence. The compiler inserts a return mechanism at the end of the last section in the OUTPUT PROCEDURE; when control passes the last statement in the OUTPUT PROCEDURE, the return mechanism terminates the merge, and passes control to the next executable statement after the MERGE statement. The RETURN statement in the OUTPUT PROCEDURE is a request for the next record (see "RETURN Statement" on page 374).

Note: The OUTPUT PROCEDURE phrase is similar to a basic PERFORM statement. For example, if you name a procedure in an OUTPUT PROCEDURE phrase, that procedure is executed during the merging operation just as if it were named in a PERFORM statement. As with the PERFORM statement, execution of the procedure ends after the last statement executes. The last statement in an output procedure can be the EXIT statement (see "EXIT Statement" on page 297).

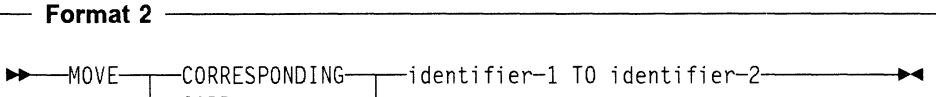
MOVE Statement

The MOVE statement transfers data from one area of storage to one or more other areas.

Format 1



Format 2



identifier-1, literal-1
Sending item

identifier-2
Receiving item or items

When Format 1 is specified, all identifiers may be either group or elementary items. The data in the sending item is moved into the data item referenced by each identifier-2 in the order in which it is specified. See "Elementary Moves" on page 323 and "Group Moves" on page 325.

When Format 2 is specified, both identifiers must be group items. CORR is an abbreviation for, and is equivalent to, CORRESPONDING.

When CORRESPONDING is specified, selected items in identifier-1 are moved to identifier-2, according to the rules for the CORRESPONDING phrase on page 209. The results are the same as if each pair of CORRESPONDING identifiers were referenced in a separate MOVE statement.

An index data item cannot be specified in a MOVE statement.

IBM Extension

You cannot specify a pointer data item in a MOVE statement. To move an address into a pointer data item, use the SET statement.

A pointer data item can be part of a group that is referred to in a MOVE statement.

A pointer move occurs when:

- The sending or receiving item of a MOVE statement contains a pointer,
- Both items are at least 16 bytes long and properly aligned, and,

- Both are alphanumeric or group items.

If the items being moved are 01-level items, or part of a 01-level structure, they must be at the same offset relative to a 16-byte boundary. All 01-level items in Working-storage are aligned on 16-byte boundaries.

For more information about pointer alignment, see "Pointer Alignment" on page 176.

A pointer data item can be part of a group that is referred to in a MOVE CORRESPONDING statement; however, movement of the pointer data item will not take place.

End of IBM Extension

The evaluation of the length of the sending or receiving item may be affected by the DEPENDING ON phrase of the OCCURS clause (see "OCCURS Clause" on page 134).

Any length evaluation, subscripting, or reference modification associated with the sending item is evaluated only once, immediately before the data is moved to the first of the receiving items. Any length evaluation, subscripting, or reference modification associated with a receiving item is evaluated immediately before the data is moved into it.

For example, 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 defined as an intermediate result item. The subscript B has changed in value between the time that the first move took place and the time that the final move to C(B) is executed.

After execution of a MOVE statement, the sending item contains the same data as before execution (unless a receiving item overlaps the sending item, in which case the contents are not predictable).

Unexpected results can occur when a redefining item is moved to the redefined item (that is, if B REDEFINES C and the statement MOVE B TO C is executed). Unexpected results can also occur when a redefined item is moved to an item redefining it (from the previous example, unexpected results occur if the statement MOVE C TO B is executed).

Elementary Moves

An elementary move is one in which the receiving item is an elementary item, and the sending item is an elementary item or a literal. Any necessary conversion of data from one form of internal representation to another takes place during the move, along with any specified editing in, or de-editing implied by, the receiving item.

De-editing is the logical removal of all editing characters from a numeric-edited data item in order to determine that item's unedited numeric value.

Each elementary item belongs to one of the following categories:

Alphabetic—includes alphabetic data items and the figurative constant SPACE.

Alphanumeric—includes alphanumeric data items, nonnumeric literals, and all figurative constants except SPACE. (The figurative constant ZERO is alphanumeric only when it is moved to an alphanumeric or alphanumeric-edited item.)

Alphanumeric-edited—includes alphanumeric-edited data items.

Numeric—includes numeric data items, numeric literals, and the figurative constant ZERO. (The figurative constant ZERO is numeric only when it is moved to a numeric or numeric-edited item.)

Numeric-edited—includes numeric-edited data items.

IBM Extension

Boolean—includes Boolean data items and Boolean literals.

End of IBM Extension

The following rules outline the execution of valid elementary moves. When the receiving item is:

Alphabetic:

- Alignment and any necessary space filling occur as described under "Alignment Rules" on page 107.
- If the size of the sending item is greater than the size of the receiving item, excess characters on the right are truncated after the receiving item is filled.

Alphanumeric or **Alphanumeric-edited**:

- Alignment and any necessary space filling take place, as described under "Alignment Rules" on page 107.
- If the size of the sending item is greater than the size of the receiving item, excess characters on the right are truncated after the receiving item is filled.
- If the sending item has an operational sign, the absolute value is used. If the operational sign occupies a separate character, that character is not moved, and the size of the sending item is considered to be one less character than the actual size.

IBM Extension

- If the sending item is Boolean, the data is moved as if the sending item were described as an alphanumeric item of length 1.

End of IBM Extension

Numeric or **Numeric-edited**:

- Except where zeros are replaced because of editing requirements, alignment by decimal point and any necessary zero filling take place, as described under "Alignment Rules" on page 107.
- If the receiving item is signed, the sign of the sending item is placed in the receiving item, with any necessary sign conversion. If the sending item is unsigned, a positive operational sign is generated for the receiving item.
- If the receiving item is unsigned, the absolute value of the sending item is moved, and no operational sign is generated for the receiving item.
- When the sending item is alphanumeric, the data is moved as if the sending item were described as an unsigned integer.
- De-editing allows the moving of a numeric-edited data item into a numeric or numeric-edited receiver. The compiler accomplishes this by first establishing the unedited value of the numeric-edited item (this value can be signed), then moving the unedited numeric value to the receiving numeric or numeric-edited data item.

  For more information, see the *COBOL/400 User's Guide*.

IBM Extension

Boolean:

- For a Boolean receiving item, only the first byte of the sending item is moved.
- If the sending item is alphanumeric, the first character of the sending item is moved. The characters "0" and "1" are equivalent to the Boolean values B"0" and B"1", respectively.
- If the sending item is zero, it is treated as the Boolean literal B"0".

End of IBM Extension

Notes:

1. If the receiving item is alphanumeric or numeric-edited, and the sending item is a scaled integer (that is, has a P as the rightmost character in its PICTURE character-string), the scaling positions are treated as trailing zeros when the MOVE statement is executed.
2. If the receiving item is numeric and the sending item is alphanumeric literal or ALL literal, then all characters of the literal must be numeric characters.

Table 35 shows valid and invalid elementary moves for each category. In the figure:

- YES = Move is valid.
- NO = Move is invalid.

Table 35. Valid and Invalid Elementary Moves

Sending Item Category	Receiving Item Category						
	Alphabetic	**Alphanumeric**	**Alphanumeric-edited**	**Numeric Integer**	**Numeric Noninteger**	**Numeric-edited**	**BOOLEAN[6]**
Alphabetic and SPACE	YES	YES	YES	NO	NO	NO	NO
Alphanumeric	YES	YES	YES	YES[4]	YES[4]	YES[4]	YES[5]
Nonnumeric Literal	YES	YES	YES	YES[1]	YES[1]	YES[1]	YES[5]
Alphanumeric-edited	YES	YES	YES	NO	NO	NO	NO
Numeric Integer[2]	NO	YES	YES	YES	YES	YES	NO
Numeric Noninteger[2]	NO	NO	NO	YES	YES	YES	NO
Numeric-edited	NO	YES	YES	YES	YES	YES	NO
LOW/HIGH-VALUE QUOTES	NO	YES	YES	NO	NO	NO	NO
ZERO	NO	YES	YES	YES	YES	YES	YES
BOOLEAN[3 6]	NO	YES	YES	NO	NO	NO	YES

Notes to Table 35:

1. Nonnumeric literal must consist only of numeric characters
2. Includes numeric literals
3. Includes Boolean literals
4. Move proceeds as if sending item were an unsigned integer
5. First character of sending item is moved, regardless of its value
6. Boolean items are an IBM Extension.

Group Moves

A group move is one in which one or both of the sending and receiving items are group items. A group move is treated exactly as though it were an alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In a group move, the receiving item is filled without consideration for the individual elementary items contained within either the sending item or the receiving item. **All** group moves are valid.

De-editing does not occur for group moves.

IBM Extension

WHEN-COMPILED Special Register

WHEN-COMPILED is a 16-character alphanumeric data item with the format:

```
MM/DD/YYhh.mm.ss (MONTH/DAY/YEARhour.minute.second)
```

WHEN-COMPILED makes available to the object program the date-and-time-compiled constant carried in the object module at the start of compilation; it is valid only as the sending item in a MOVE statement. You can move WHEN-COMPILED special register data to an alphanumeric-edited item.

For example, if compilation began at 2:04 PM on 15 May 1986, WHEN-COMPILED would contain the value 05/15/8614.04.00.

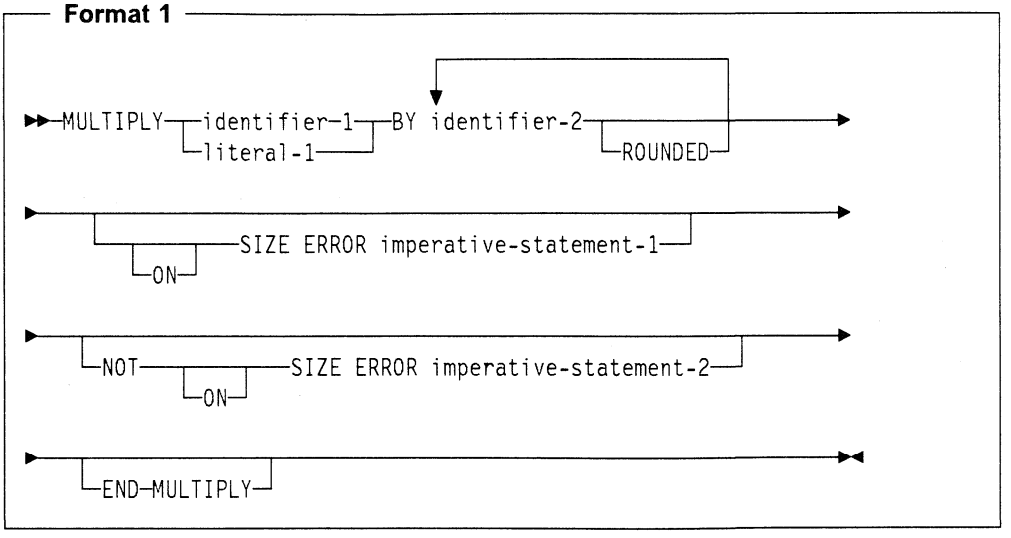
The TIMSEP parameter of job-related commands (such as CHGJOB) specifies the time-separation character used in the WHEN-COMPILED special register.

End of IBM Extension

MULTIPLY Statement

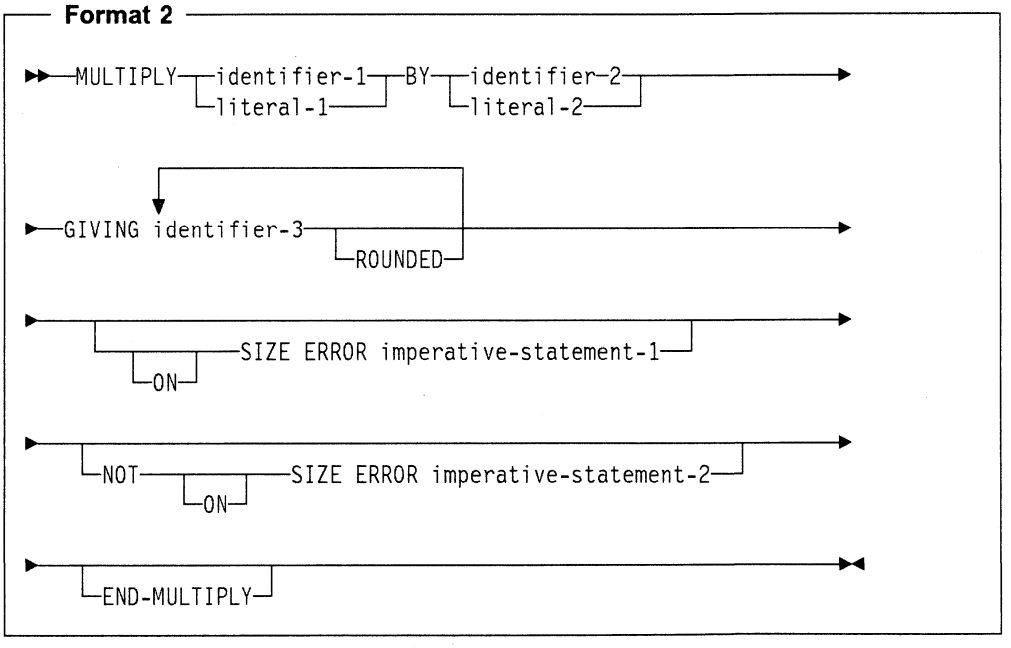
The MULTIPLY statement multiplies numeric items and sets the values of data items equal to the results.

Format 1



In Format 1, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2; the product is then placed in identifier-2. This process is repeated for each successive occurrence of identifier-2, in the left-to-right order in which identifier-2 is specified.

Format 2



In Format 2, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2 or literal-2. The product is then stored in the data item(s) referenced by identifier-3.

For all Formats:

identifier-1, identifier-2
Must name an elementary numeric item.

literal
Must be a numeric literal.

For Format-2:

identifier-3
Must name an elementary numeric or numeric-edited item.

The composite of operands must not contain more than 18 digits.

IBM Extension

The composite of all operands in an arithmetic statement can have a maximum length of 30 digits.

End of IBM Extension

ROUNDED Phrase

For Formats 1 and 2, see "ROUNDED Phrase" on page 210.

SIZE ERROR Phrases

For Formats 1 and 2, see "SIZE ERROR Phrases" on page 210.

END-MULTIPLY Phrase

This explicit scope terminator serves to delimit the scope of the MULTIPLY statement. END-MULTIPLY permits a conditional MULTIPLY statement to be nested in another conditional statement. END-MULTIPLY may also be used with an imperative MULTIPLY statement.

For more information, see "Delimited Scope Statements" on page 207.

OPEN Statement

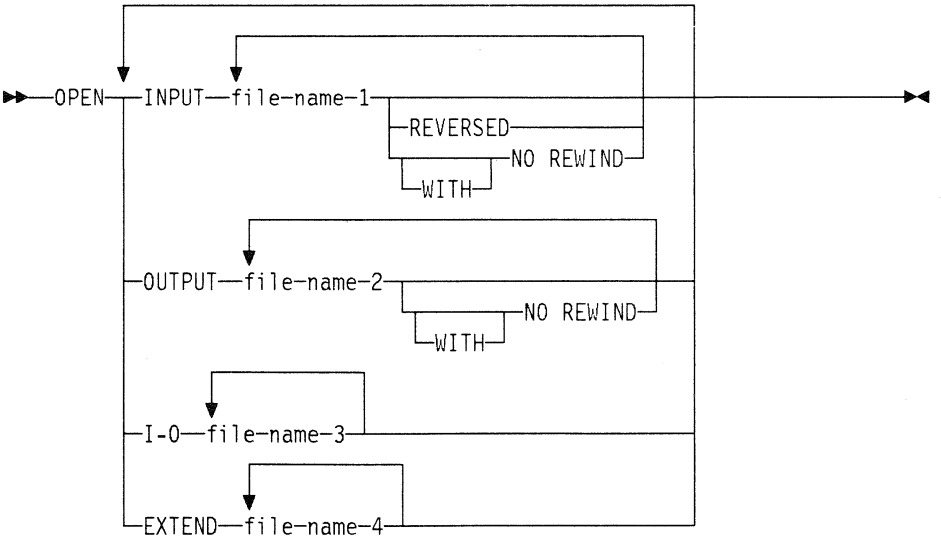
The OPEN statement initiates the processing of files. It also checks and/or writes labels.

The OPEN statement varies depending on the type of file:

- Sequential files
- Indexed files
- Relative files
- Transaction files.

Sequential Files

Format 1—Sequential



At least one of the phrases, INPUT, OUTPUT, I-O, or EXTEND, must be specified with the OPEN keyword. The INPUT, OUTPUT, I-O, and EXTEND phrases may appear in any order.

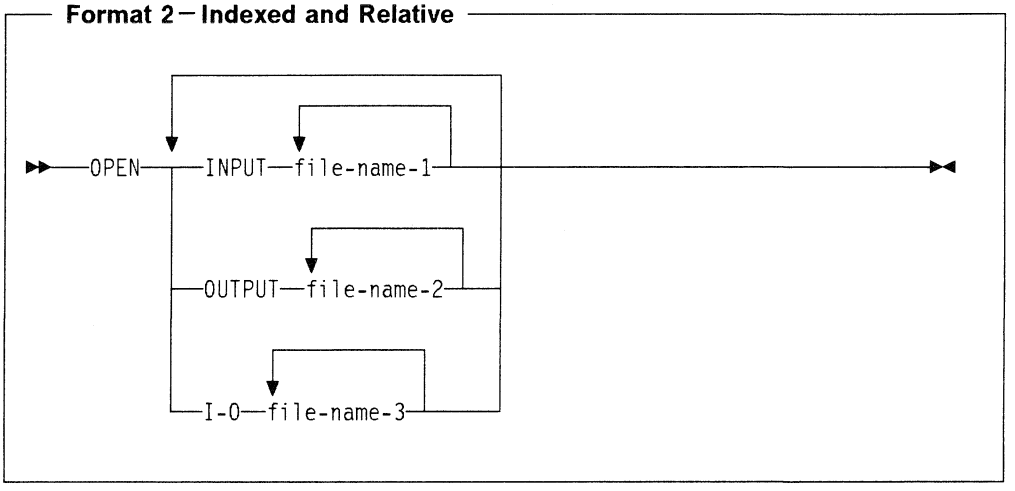
file-name

Designates a file upon which the OPEN statement is to operate. If more than one file is specified, the files need not have the same organization or access. Each file-name must be defined in an FD entry in the Data Division, and must not name a sort or merge file. The FD entry must be equivalent to the information supplied when the file was defined.

See Table 36 on page 331 for an illustration of sequential organization, access and device considerations for the OPEN statement.

Indexed and Relative Files

Format 2—Indexed and Relative



At least one of the phrases, INPUT, OUTPUT, or I-O, must be specified with the OPEN keyword. The INPUT, OUTPUT, and I-O phrases may appear in any order.

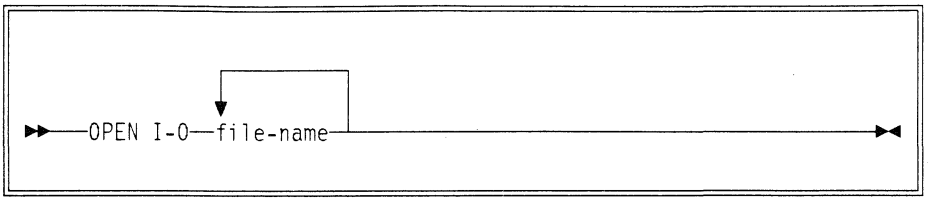
file-name

Designates a file upon which the OPEN statement is to operate. If more than one file is specified, the files need not have the same organization or access. Each file-name must be defined in an FD entry in the Data Division, and must not name a sort or merge file. The FD entry must be equivalent to the information supplied when the file was defined.

See Table 37 on page 331 and Table 38 on page 332 for an illustration of relative and indexed organization, access and device considerations for the OPEN statement.

Transaction Files

Format 3—TRANSACTION



A TRANSACTION file must be opened in the I-O mode.

The OPEN statement can cause a program device to be implicitly acquired for a TRANSACTION file. For a further discussion about the acquiring of program devices, see the "ACQUIRE Statement" on page 244.

OPEN Statement Considerations

Table 36. Sequential Organization

ACCESS	SEQUENTIAL					
DEVICE	**PRINTER**	**TAPEFILE**	**DISKETTE**	**DISK**	**DATABASE**	**FORMATFILE**
OPEN Verb	S	L,S	S	S	S,C,X	S
INPUT	–	O,A,F,L1,N1,V	O,A,F,N1,V	O,A,K,F,N1	O,A,K,F,N1	O,A,K,F
OUTPUT	R,J,U	O,J,L2,N1,U	O,J,N1,U	O,G,N1,U	O,G,N1	O,G
I—O	–	–	–	O,M,K,T	O,M,K	O,M,K
NO REWIND	–	O,D	–	–	–	–
REVERSED	–	O,B	–	–	–	–
EXTEND	–	O,E,L3,T,W	O,W	O,E,T	O,E	–

Table 37. Relative Organization

Device	DISK			DATABASE		
Access	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**
OPEN Verb	H,S	H,S	H,S	H,S,C,X	H,S,C,X	H,S,C,X
INPUT	O,A,K,N1	O,A	O,A,K,N2	O,A,K,N1	O,A	O,A,K,N2
OUTPUT	O,G,U	O,G,U	O,G,U	O,G	O,G	O,G
I—O	O,M,K	O,M	O,M,K	O,M,K	O,M	O,M,K
NO REWIND	–	–	–	–	–	–
REVERSED	–	–	–	–	–	–
EXTEND	–	–	–	–	–	–

Table 38. Indexed Organization

Device	DISK			DATABASE		
Access	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**
OPEN Verb	S	S	S	S,C,X	S,C,X	S,C,X
INPUT	O,A,K,N1	O,A	O,A,K,N2	O,A,K,N1	O,A	O,A,K,N2
OUTPUT	O,G,N1,U	O,G,U	O,G,U,N2	O,G,N1	O,G	O,G,N2
I–O	O,M,K	O,M	O,M,K	O,M,K	O,M	O,M,K
NO REWIND	–	–	–	–	–	–
REVERSED	–	–	–	–	–	–
EXTEND	–	–	–	–	–	–

Letter Code **Meaning**

– Combination is not valid.

A The file is opened for input operations. The file position indicator is set to the first record in the file. If no records exist in the file, the file position indicator is set so that processing of the first sequential READ statement results in an AT END condition.

B OPEN statement processing positions the file at its end. Subsequent READ statements make the data records available in reverse order, starting with the last record. REVERSED can only be specified for input files.

The REVERSED phrase is an obsolete element and is to be deleted from the next revision of the ANSI Standard.

If the concept of reels has no meaning for the storage medium (for example, a direct access device), the REVERSED and NO REWIND phrases do not apply. When the phrases are used in this situation, a file status of 07 is set for NO REWIND, and a file status of 00 is set for REVERSED.

IBM Extension

C The file may be placed under commitment control.

See "Commitment Control" in the *COBOL/400 User's Guide* for more information.

End of IBM Extension

D The OPEN statement does not reposition the file. The tape must be positioned at the beginning of the desired file before processing of the OPEN statement.

If the concept of reels has no meaning for the storage medium (for example, a direct access device), the REVERSED and NO REWIND phrases do not apply. When the phrases are used in this situation, a file status of 07 is set.

IBM Extension

The system keeps track of the current position on the tape and automatically positions the tape to the proper place. When processing a multi-file tape volume, all CLOSE statements should specify the NO REWIND phrase. When the next file on the volume is opened, the system determines which direction the tape should be moved to most efficiently get to the desired file.

End of IBM Extension

E The EXTEND phrase permits opening the file for output operations. OPEN EXTEND statement processing prepares the file for the addition of records. These additional records immediately follow the last record in the file. Subsequent WRITE statements add records as if the file had been opened for OUTPUT. The EXTEND phrase can be specified when a file is being created.

The EXTEND phrase is not allowed for files for which the LINAGE clause has been specified.

F If SELECT OPTIONAL is specified in the file-control entry, OPEN statement processing causes the program to check for the presence or absence of this file at run time. If the file is absent, the first READ statement for this file causes the AT END condition to occur.

IBM Extension

G Only a physical file is cleared when opened for OUTPUT. When the file is successfully opened, it contains no records. If an attempt is made to open a logical file for OUTPUT, the file is opened but no records are deleted. The file is treated as though the EXTEND phrase had been specified. To clear a logical file, all the members on which the logical file is based should be cleared.

End of IBM Extension

H Not allowed for logical file members:

- That are based on more than one physical file.
- That contain select/omit logic.

J The file is opened to allow only output operations. When the file is successfully opened, it contains no records.

IBM Extension

K The first record to be made available to the program can be specified at run time by using the POSITION parameter on the OVRDBF CL command. See the *CL Reference* for more information on this command.

End of IBM Extension

L When label records are specified but not present, or when label records are present but not specified, processing of the OPEN statement can have unpredictable results.

L1 The beginning labels are checked.

L2 The labels are checked, then new labels are written.

L3 The following results occur:

- Beginning file labels are processed only if this is a single-volume file.
- Beginning volume labels of the last existing volume are checked.
- The file is positioned to the existing ending file labels. The labels are checked and then deleted.
- Processing continues as if the file were opened as an output file.

M The file is opened for both input and output operations. The file position indicator is set to the first record in the file. If no records exist in the file, the file position indicator is set so that processing of the first sequential READ statement results in an AT END condition.

N1 Under GENOPT option *NOBLK, the compiler generates code to block output records and unblock input records if the following conditions are satisfied:

- The file access is sequential.
- The organization of the file is sequential or indexed, and the file is open only for input or output; or the organization of the file is relative, and the file is open only for input.
- The file is assigned to DISK, DATABASE, DISKETTE, or TAPEFILE.
- No START statements are specified for the file.

The BLOCK CONTAINS clause does not control the blocking factor for any files except tape files.

START statements are allowed if you specify both GENOPT option *BLK and the BLOCK CONTAINS clause.

N2 Also, under GENOPT option *BLK, the BLOCK CONTAINS clause causes the compiler to generate code that blocks output records and unblocks input records if the following conditions are satisfied:

- The file access is dynamic.
- The organization of the file is sequential or indexed, and the file is open only for input or output; or the organization of the file is relative, and the file is open only for input.
- The file is assigned to DISK or DATABASE.

If the BLOCK CONTAINS clause specifies a record size of zero, the system default blocking factor applies.

O You can specify this combination.

R You must specify this combination for the device shown.

S The successful execution of an OPEN statement determines the availability of the file and results in that file being in an open mode. The file is unavailable if the OPEN operation fails. A file is available if it is physically present and is recognized by the input-output control system. Table 39 on page 336 shows the results of opening available and unavailable files.

T If the OPTIONAL phrase is used in the SELECT clause for the file, and GENOPT(*NOCRTF) is specified in the CRTCBLPGM command, a compile-time error will occur. If the OPTIONAL phrase is used, and if GENOPT(*CRTF) is specified in the CRTCBLPGM command, the file will be created by the OPEN operation if it is unavailable. Tape files must be overridden to physical files for dynamic file creation. For dynamic file creation, specify GENOPT(*CRTF) on the CRTCBLPGM command. If the OPTIONAL phrase is not specified, file status 90 will be returned.

Note: The maximum record length for a file that can be created dynamically is 32 766.

U If the file is unavailable, it will not be created by the OPEN operation if GENOPT(*NOCRTF) is specified in the CRTCBLPGM command. For dynamic file creation, specify GENOPT(*CRTF) in the CRTCBLPGM command. (*NOCRTF is the default unless otherwise specified.) Device files must be overridden to physical files for dynamic file creation.

Note: The maximum record length for a file that can be created dynamically is 32 766.

V If SELECT OPTIONAL is specified in the file-control entry and GENOPT(*CRTF) is specified in the CRTCBLPGM command, this combination is not valid.

W If SELECT OPTIONAL is specified in the file-control entry and GENOPT(*CRTF) is specified in the CRTCBLPGM command, this combination is valid.

X If the file contains null-capable fields, a file status of 0P is set.

OPEN Statement Notes

Table 39. Availability of a File

	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
I-O	Normal open	Open is unsuccessful
I-O (optional file)	Normal open	Open causes the file to be created[1]
OUTPUT	Normal open; the file contains no records	Open causes the file to be created[1]
EXTEND	Normal open	Open is unsuccessful
EXTEND (optional file)	Normal open	Open causes the file to be created[1]

Note: [1] If GENOPT(*CRTF) has been specified, and a library-name has not been provided, the file will be created in the library designated by *CURLIB. If *CURLIB has not been defined, the file will be created in the library QTEMP. For further information about library names, see the CRTCBLPGM CL command description in the *COBOL/400 User's Guide*.

1. The successful execution of the OPEN statement makes the associated record area available to the program; it does not obtain or release the first data record.
2. An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements, except a SORT or MERGE statement with the USING or GIVING phrase.

   The following table shows the permissible input-output statements for sequential files. An 'X' indicates that the specified statement may be used with the open mode given at the top of the column.

Table 40. Permissible Statements for Sequential Files

	Open Mode			
Statement	Input	Output	I-O	Extend
READ	X		X	
WRITE		X		X
REWRITE			X	

The following table shows the permissible statements for indexed and relative files. An 'X' indicates that the specified statement, used in the access mode given for that row, may be used with the open mode given at the top of the column.

Table 41. Permissible Statements for Indexed and Relative Files

File Access Mode	Statement	Open Mode			
		Input	Output	I-O	Extend
Sequential	READ	X		X	
	WRITE		X		
	REWRITE			X	
	START	X		X	
	DELETE			X	
Random	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START				
	DELETE			X	
Dynamic	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START	X		X	
	DELETE			X	

3. A file may be opened for INPUT, OUTPUT, I-O, or EXTEND (sequential files only) in the same program. After the first OPEN statement execution for a given file, each subsequent OPEN statement execution must be preceded by a successful CLOSE file statement execution without the REEL or UNIT phrase (for sequential files only), or the LOCK phrase.
4. If the FILE STATUS clause is specified in the FILE-CONTROL entry, the associated status key is updated when the OPEN statement is executed. For more information about the status key, refer to "Common Processing Facilities" on page 214.
5. If an OPEN statement is issued for a file already in the open status, the EXCEPTION/ERROR procedure (if specified) for this file is executed and file status 41 is returned.

PERFORM Statement

The PERFORM statement transfers control explicitly to one or more procedures and implicitly returns control to the next executable statement after execution of the specified procedure(s) or imperative statements is completed.

The PERFORM statement can be:

An out-of-line PERFORM statement
Procedure-name-1 is specified.

An in-line PERFORM statement
Procedure-name-1 is omitted.

An in-line PERFORM must be delimited by the END-PERFORM phrase.

The in-line and out-of-line formats cannot be combined. For example, if procedure-name-1 is specified, the imperative-statement and the END-PERFORM phrase must not be specified.

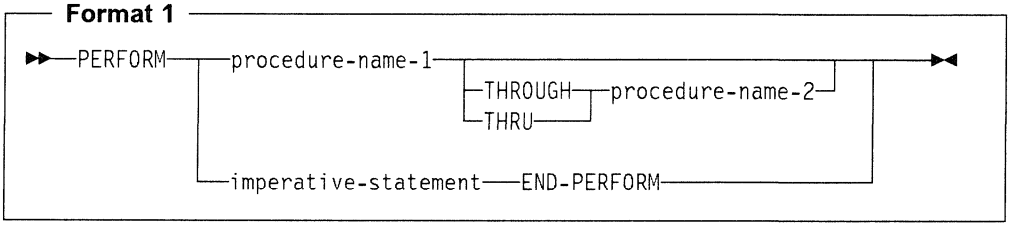
There are four PERFORM statement formats:

- Basic PERFORM
- TIMES phrase PERFORM
- UNTIL phrase PERFORM
- VARYING phrase PERFORM.

Basic PERFORM Statement

The procedure(s) referenced in the basic PERFORM statement are executed once, and control then passes to the next executable statement following the PERFORM statement.

Format 1



procedure-name
Must name a section or paragraph in the Procedure Division.

When both procedure-name-1 and procedure-name-2 are specified, if either is a procedure-name in a declarative procedure, both must be procedure-names in the same declarative procedure.

If procedure-name-1 is specified, imperative-statement and the END-PERFORM phrase must not be specified.

If procedure-name-1 is omitted, imperative-statement and the END-PERFORM phrase must be specified.

imperative-statement
The statement(s) to be executed for an in-line PERFORM.

END-PERFORM
Delimits the scope of the in-line PERFORM statement. Execution of an in-line PERFORM is completed after the last statement contained within it has been executed.

An in-line PERFORM statement functions according to the same general rules as an otherwise identical out-of-line PERFORM statement, except that statements contained within the in-line PERFORM are executed in place of the statements contained within the range of procedure-name-1 (through procedure-name-2, if specified). Unless specifically qualified by the word **in-line** or **out-of-line**, all the rules that apply to the out-of-line PERFORM statement also apply to the in-line PERFORM.

Whenever an out-of-line PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1. Control is always returned to the statement following the PERFORM statement. The point from which this control is returned is determined as follows:

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

The only necessary relationship between procedure-name-1 and procedure-name-2 is that a consecutive sequence of operations is executed, beginning at the procedure named by procedure-name-1 and ending with the execution of the procedure named by procedure-name-2.

PERFORM statements may be specified within the performed procedure. If there are two or more logical paths to the return point, then procedure-name-2 may name a paragraph that consists only of an EXIT statement; all the paths to the return point must then lead to this paragraph.

When both procedure-name-1 and procedure-name-2 are specified, GO TO and PERFORM statements can appear within the sequence of statements contained in these paragraphs or sections. A GO TO statement should not refer to a procedure-name outside the range of procedure-name-1 through procedure-name-2. If this is done, results are unpredictable and are not diagnosed.

When only procedure-name-1 is specified, PERFORM and GO TO statements can appear within the procedure. A GO TO statement should not refer to a procedure-name outside the range of procedure-name-1. If this is done, results are unpredictable and are not diagnosed.

When the performed procedures include another PERFORM statement, the sequence of procedures associated with the embedded PERFORM statement

must be totally included in or totally excluded from the performed procedures of the first PERFORM statement. That is, an active PERFORM statement whose execution point begins within the range of performed procedures of another active PERFORM statement must not allow control to pass through the exit point of the other active PERFORM statement. In addition, two or more such active PERFORM statements must not have a common exit.

IBM Extension

Two or more active PERFORM statements can have a common exit point.

End of IBM Extension

Figure 18 illustrates valid sequences of execution for PERFORM statements.

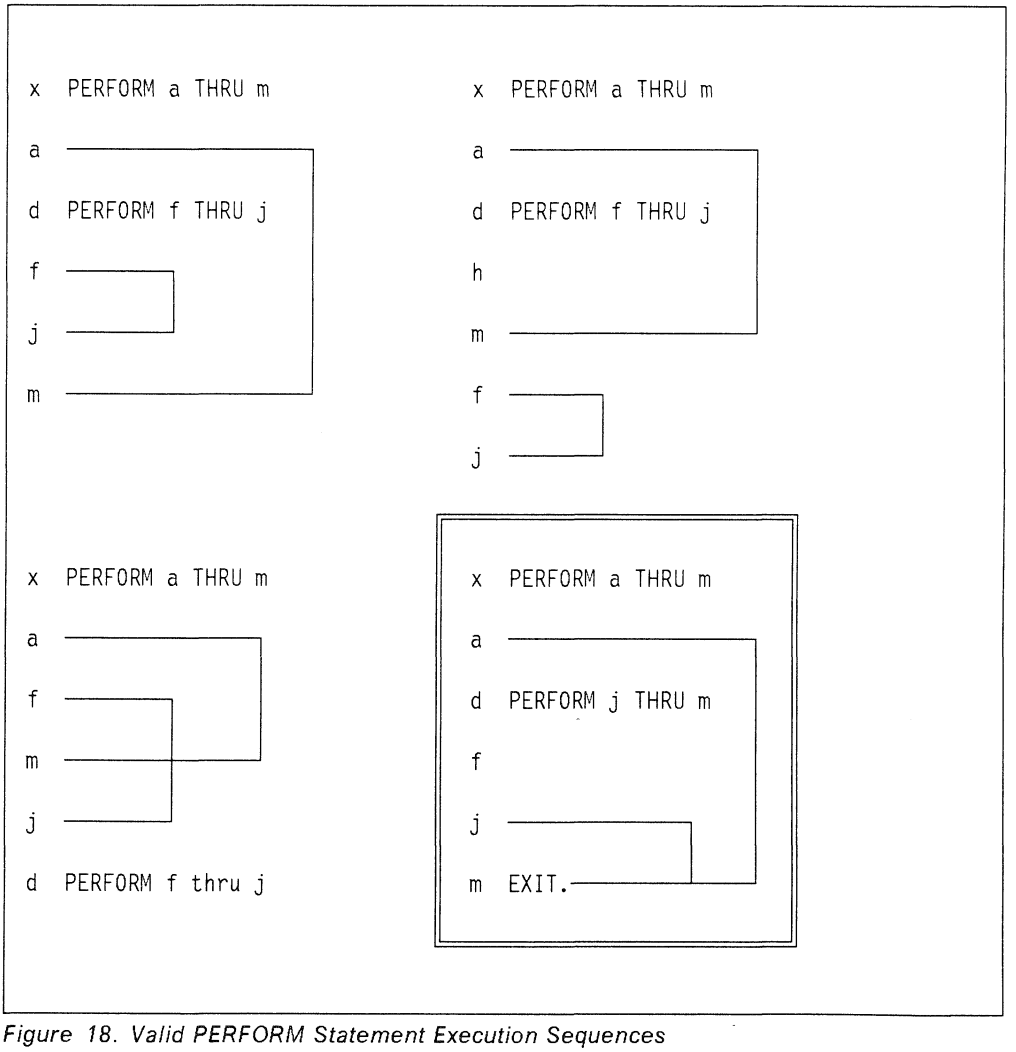


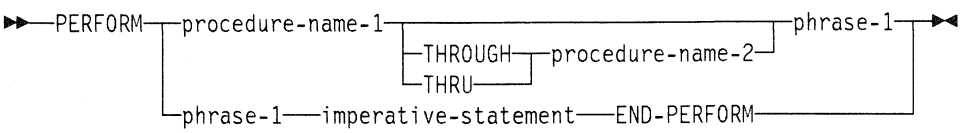
Figure 18. Valid PERFORM Statement Execution Sequences

When control passes to the sequence of procedures by means other than a PERFORM statement, control passes through the exit point to the next executable statement, as if no PERFORM statement referred to these procedures.

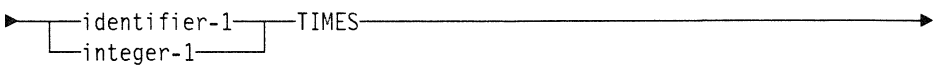
PERFORM with TIMES Phrase

The procedure(s) referred to in the TIMES phrase PERFORM statement are executed the **number of times** specified by the value in identifier-1 or integer-1. Control then passes to the next executable statement following the PERFORM statement.

Format 2



where phrase-1 **is:**



identifier-1

Must name an integer item.

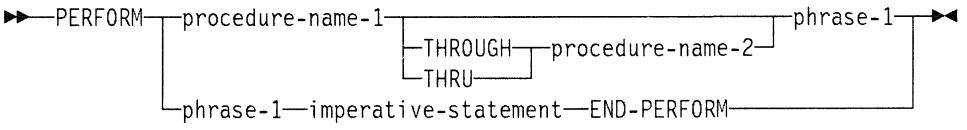
If identifier-1 is zero or a negative number at the time the PERFORM statement is initiated, control passes to the statement following the PERFORM statement.

After the PERFORM statement has been initiated, any change to identifier-1 has no effect in varying the number of times the procedures are initiated.

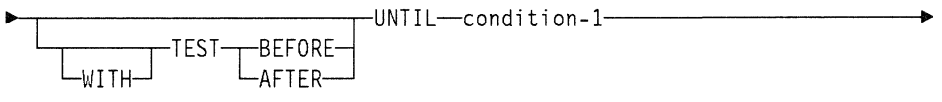
PERFORM with UNTIL Phrase

In the UNTIL phrase format, the procedure(s) referred to are performed **until** the condition specified by the UNTIL phrase is true. Control is then passed to the next executable statement following the PERFORM statement.

Format 3



where phrase-1 **is:**



condition-1

May be any condition described under "Conditional Expressions" on page 189. If the condition is true at the time the PERFORM statement is initiated, the specified procedure(s) are not executed.

Any subscripting associated with the operands specified in condition-1 is evaluated each time the condition is tested.

If the TEST BEFORE phrase is specified or assumed, the condition is tested before any statements are executed (corresponds to DO WHILE).

If the TEST AFTER phrase is specified, the statements to be performed are executed at least once before the condition is tested (corresponds to DO UNTIL).

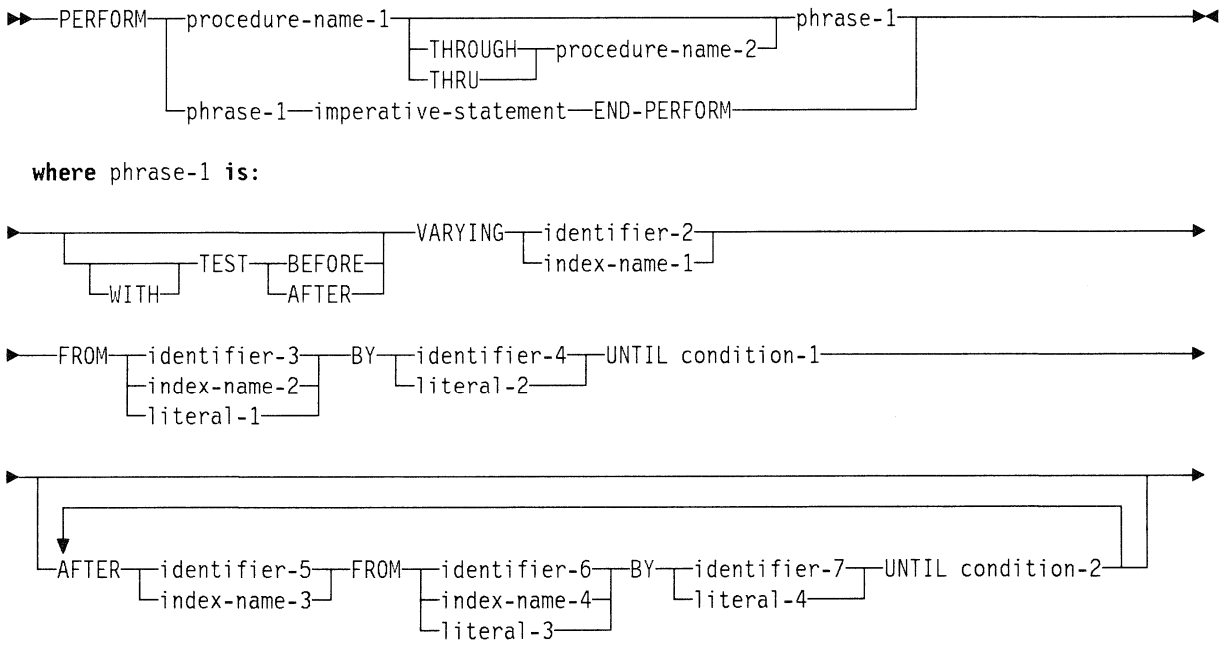
In either case, if the condition is true, control is transferred to the next executable statement following the end of the PERFORM statement. If neither the TEST BEFORE nor the TEST AFTER phrase is specified, the TEST BEFORE phrase is assumed.

PERFORM with VARYING Phrase

The VARYING phrase increases or decreases the value of one or more identifiers or index-names, according to certain rules. (See "Varying Phrase Rules" on page 350.)

The Format 4 VARYING phrase PERFORM statement can serially search an entire 7-dimensional table.

Format 4



condition-1, condition-2

May be any condition described under "Conditional Expressions" on page 189. If the condition is true at the time the PERFORM statement is initiated, the specified procedure(s) are not executed.

After the condition(s) specified in the UNTIL phrase are satisfied, control is passed to the next executable statement following the PERFORM statement.

Any subscripting associated with the operands specified in condition-1 or condition-2 is evaluated each time the condition is tested.

When TEST BEFORE is indicated, all specified conditions are tested before the first execution, and the statements to be performed are executed, if at all, only when **all** specified tests fail. When TEST AFTER is indicated, the statements to be performed are executed at least once, before any condition is tested. Any subscripting associated with the operands specified in condition-1 is evaluated each time the condition is tested.

If neither the TEST BEFORE nor the TEST AFTER phrase is specified, the TEST BEFORE phrase is assumed.

Varying Identifiers

The way in which operands are increased or decreased depends on the number of variables specified. In the following discussion, every reference to identifier-n refers equally to index-name-n (except when identifier-n is the object of the BY phrase).

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 an augmenting operation.

Varying One Identifier

Example

```
PERFORM procedure-name-1 THROUGH procedure-name-2
       VARYING identifier-2 FROM identifier-3
            BY identifier-4 UNTIL condition-1
```

1. **Identifier-2** is set equal to its starting value, identifier-3 (or literal-2).
2. **Condition-1** is evaluated as follows:
   a. If it is false, steps 3 through 5 are executed.
   b. If it is true, control passes directly to the statement following the PERFORM statement.
3. **Procedure-1** and everything up to and including **procedure-2** (if specified) are executed once.
4. **Identifier-2** is augmented by identifier-4 (or literal-4), and condition-1 is evaluated again.
5. Steps 2 through 4 are repeated until condition-1 is true.

At the end of PERFORM statement execution **identifier-2** has a value that exceeds the last-used setting by the increment/decrement value (unless condition-1 was true at the beginning of PERFORM statement execution, in which case, identifier-2 contains the current value of identifier-3).

Figure 19 illustrates the logic of the PERFORM statement when an identifier is varied with TEST BEFORE. Figure 20 illustrates the logic of the PERFORM statement when an identifier is varied with TEST AFTER.

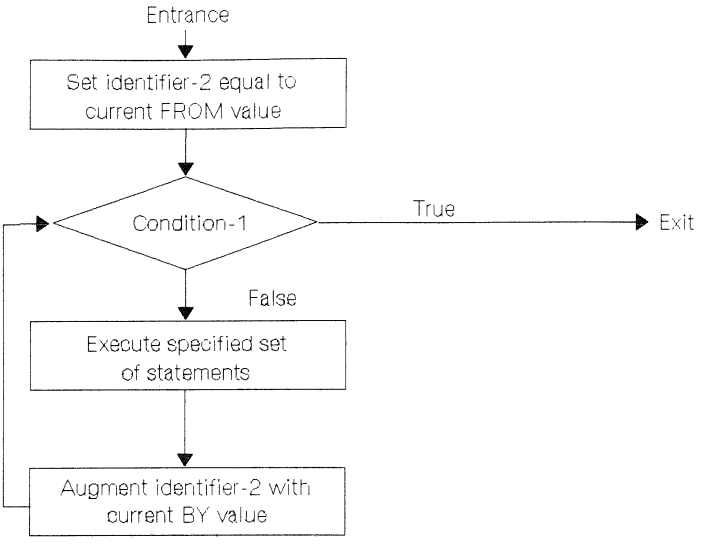


Figure 19. Varying One Identifier—with TEST BEFORE

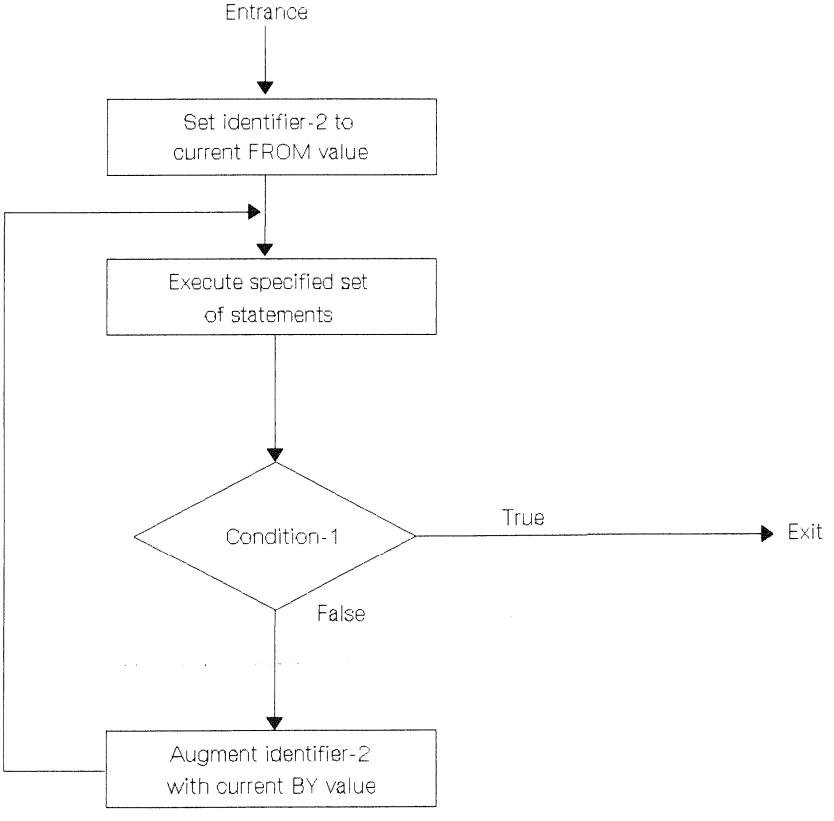


Figure 20. Varying One Identifier—with TEST AFTER

Varying Two Identifiers

Example

```
PERFORM procedure-name-1 THROUGH procedure-name-2
      VARYING identifier-2 FROM identifier-3
            BY identifier-4 UNTIL condition-1
         AFTER identifier-5 FROM identifier-6
            BY identifier-7 UNTIL condition-2
```

1. **identifier-2** and **identifier-5** are set to their initial values, identifier-3 and identifier-6, respectively.
2. **condition-1** is evaluated as follows:
   a. If it is false, steps 3 through 7 are executed.
   b. If it is true, control passes directly to the statement following the PERFORM statement.
3. **condition-2** is evaluated as follows:
   a. If it is false, steps 4 through 6 are executed.
   b. If it is true, identifier-2 is augmented by identifier-4, identifier-5 is set to the current value of identifier-6, and step 2 is repeated.
4. **procedure-name-1** and everything up to and including **procedure-name-2** (if specified) are executed once.
5. **identifier-5** is augmented by identifier-7.
6. Steps 3 through 5 are repeated until condition-2 is true.
7. Steps 2 through 6 are repeated until condition-1 is true.

At the end of PERFORM statement execution:

- **identifier-5**

  Contains the current value of identifier-6.

- **identifier-2**

  Has a value that exceeds the last-used setting by the increment/decrement value (unless condition-1 was true at the beginning of PERFORM statement execution, in which case, identifier-2 contains the current value of identifier-3).

Figure 21 illustrates the logic of the PERFORM statement when two identifiers are varied with TEST BEFORE. Figure 22 on page 347 illustrates the logic of the PERFORM statement when two identifiers are varied with TEST AFTER.

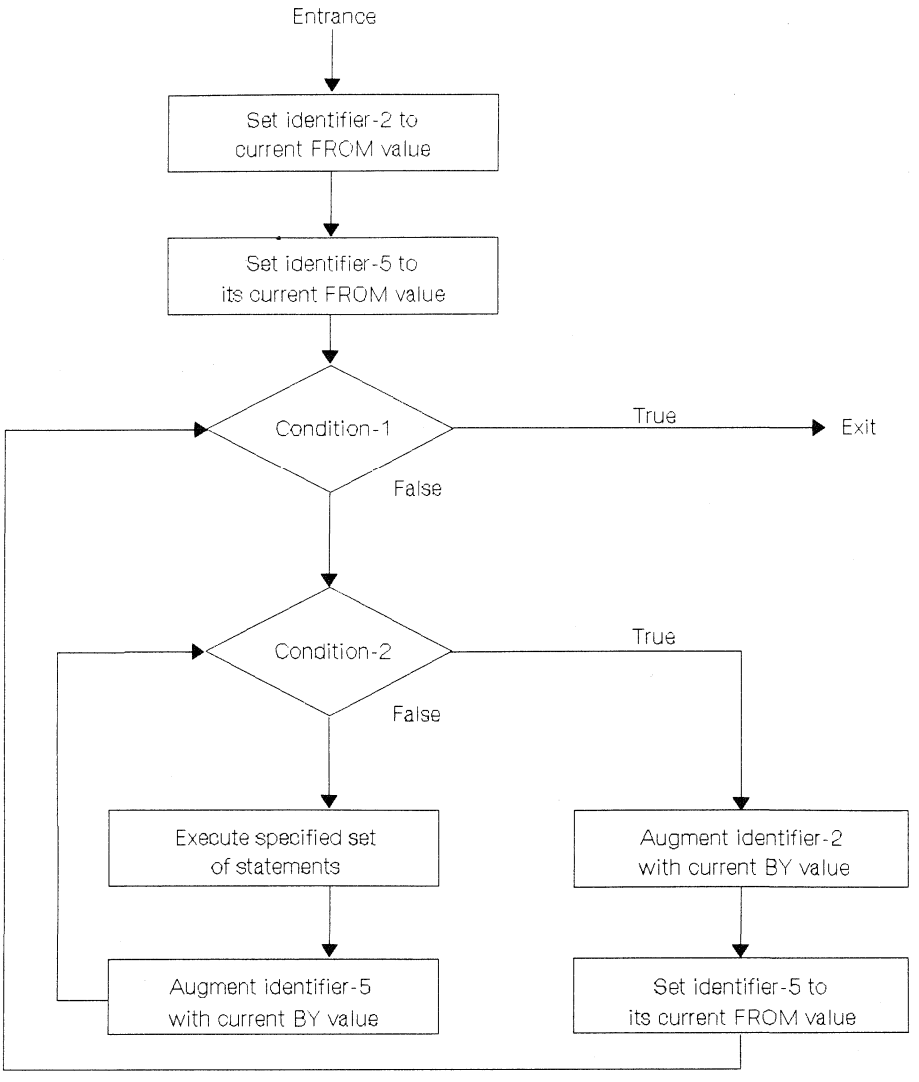


Figure 21. Varying Two Identifiers—with TEST BEFORE

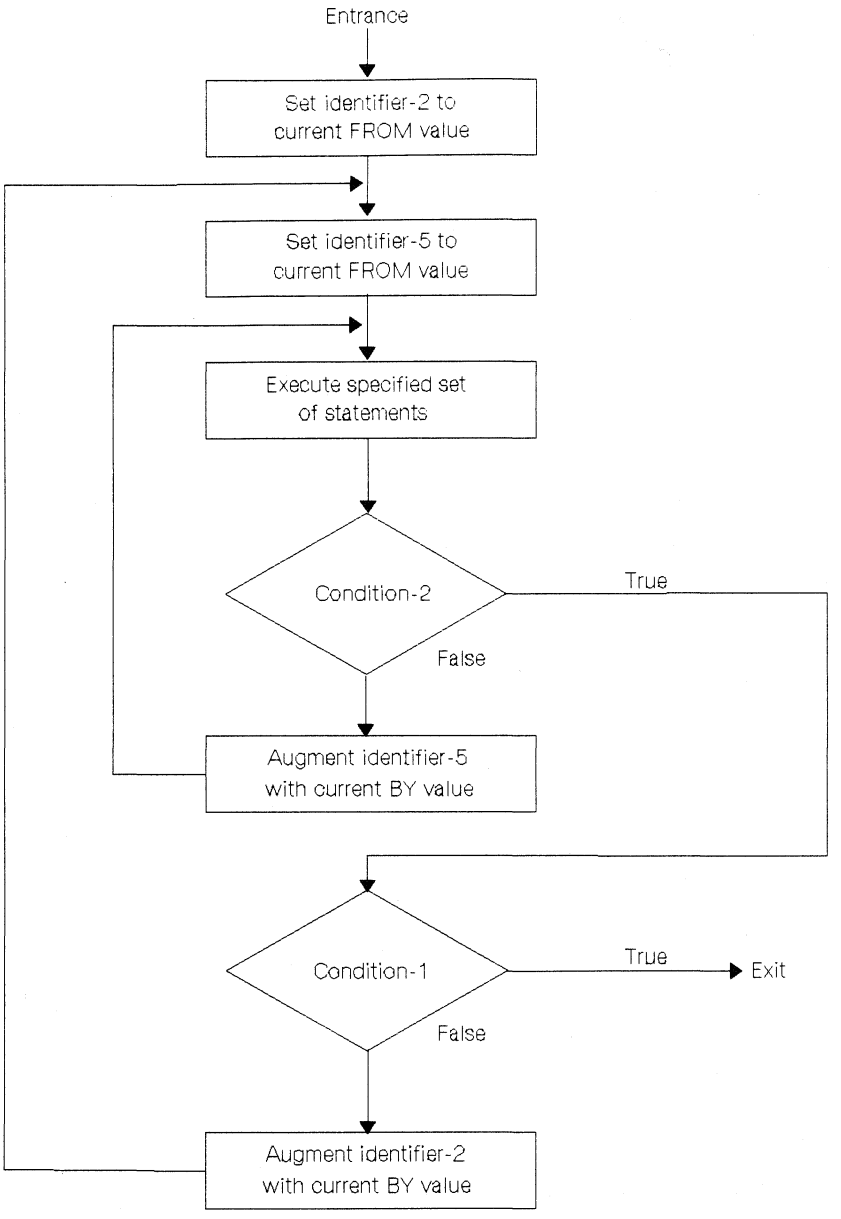


Figure 22. Varying Two Identifiers—with TEST AFTER

Varying Three Identifiers

Example

```
PERFORM procedure-name-1 THROUGH procedure-name-2
      VARYING identifier-2 FROM identifier-3
            BY identifier-4 UNTIL condition-1
         AFTER identifier-5 FROM identifier-6
            BY identifier-7 UNTIL condition-2
         AFTER identifier-8 FROM identifier-9
            BY identifier-10 UNTIL condition-3
```

The actions are the same as those for two identifiers, except that identifier-8 goes through the complete cycle each time identifier-5 is augmented by

identifier-7, which, in turn, goes through a complete cycle each time identifier-2 is varied.

At the end of PERFORM statement execution:

- **identifier-5** and **identifier-8**

  Contain the current values of identifier-6 and identifier-9, respectively.

- **identifier-2**

  Has a value exceeding its last-used setting by one increment/decrement value (unless condition-1 was true at the beginning of PERFORM statement execution, in which case, identifier-2 contains the current value of identifier-3).

Figure 23 illustrates the logic of the PERFORM statement when three identifiers are varied.

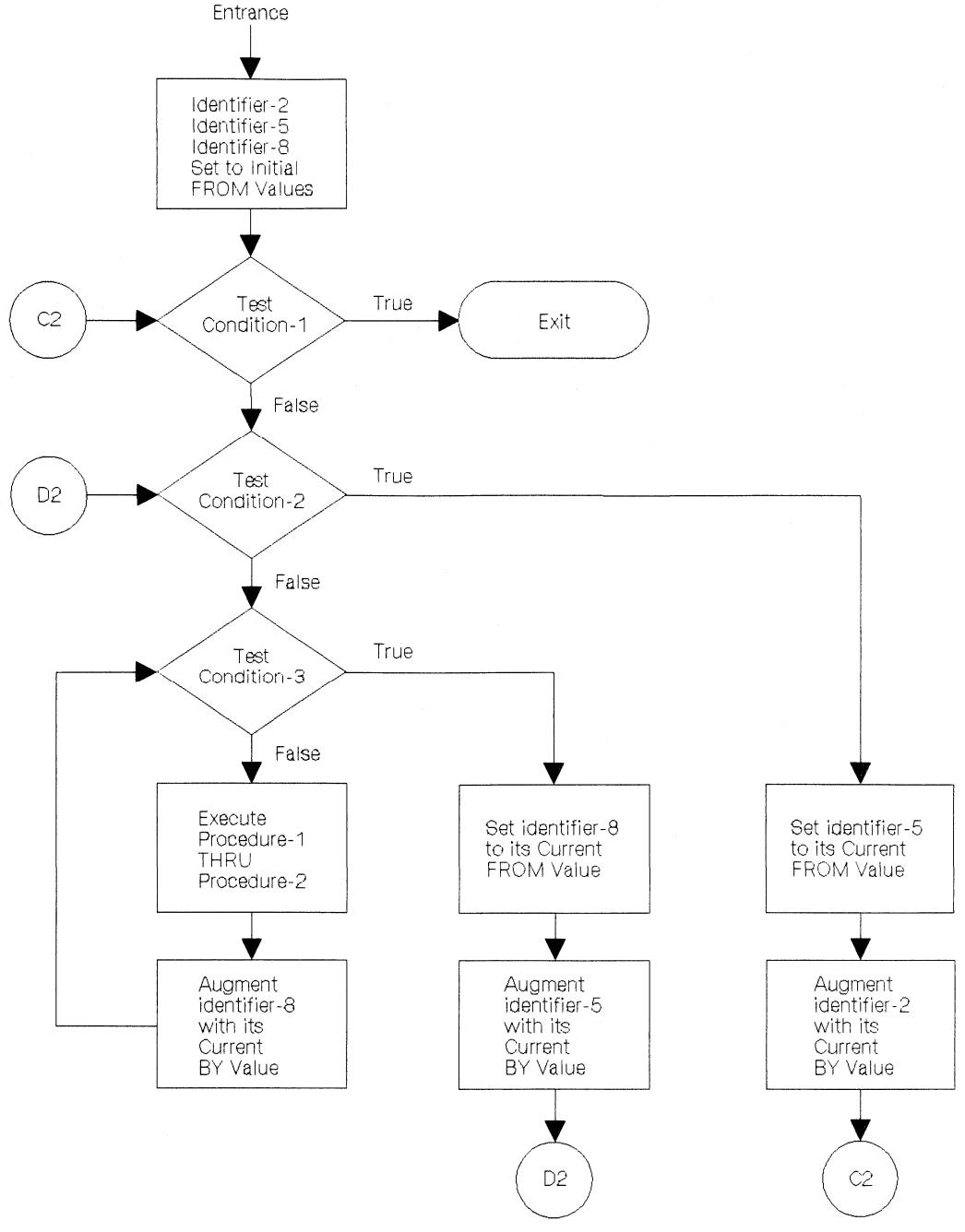


Figure 23. Format 4 PERFORM Statement Logic—Varying Three Identifiers

Varying More Than Three Identifiers

In the VARYING phrase, you may extend the examples above by adding up to four more AFTER phrases, for a total of six AFTER phrases.

Varying Phrase Rules

No matter how many variables are specified, the following rules apply:

1. In the VARYING/AFTER phrases, when an index-name is specified:
   a. The index-name is initialized and incremented or decremented according to the rules under "INDEXED BY Phrase" on page 140. (See also "SET Statement" on page 394.)
   b. In the associated FROM phrase, an identifier must be described as an integer and have a positive value; a literal must be a positive integer.
   c. In the associated BY phrase, an identifier must be described as an integer; a literal must be a nonzero integer.
2. In the FROM phrase, when an index-name is specified:
   a. In the associated VARYING/AFTER phrase, an identifier must be described as an integer. It is initialized, as described in the SET statement.
   b. In the associated BY phrase, an identifier must be described as an integer and have a nonzero value; a literal must be a nonzero integer.
3. In the BY phrase, identifiers and literals must have nonzero values.
4. Changing the values of identifiers and/or index-names in the VARYING, FROM, and BY phrases during execution changes the number of times the procedures are executed.
5. The way in which operands are incremented or decremented depends on the number of variables specified.

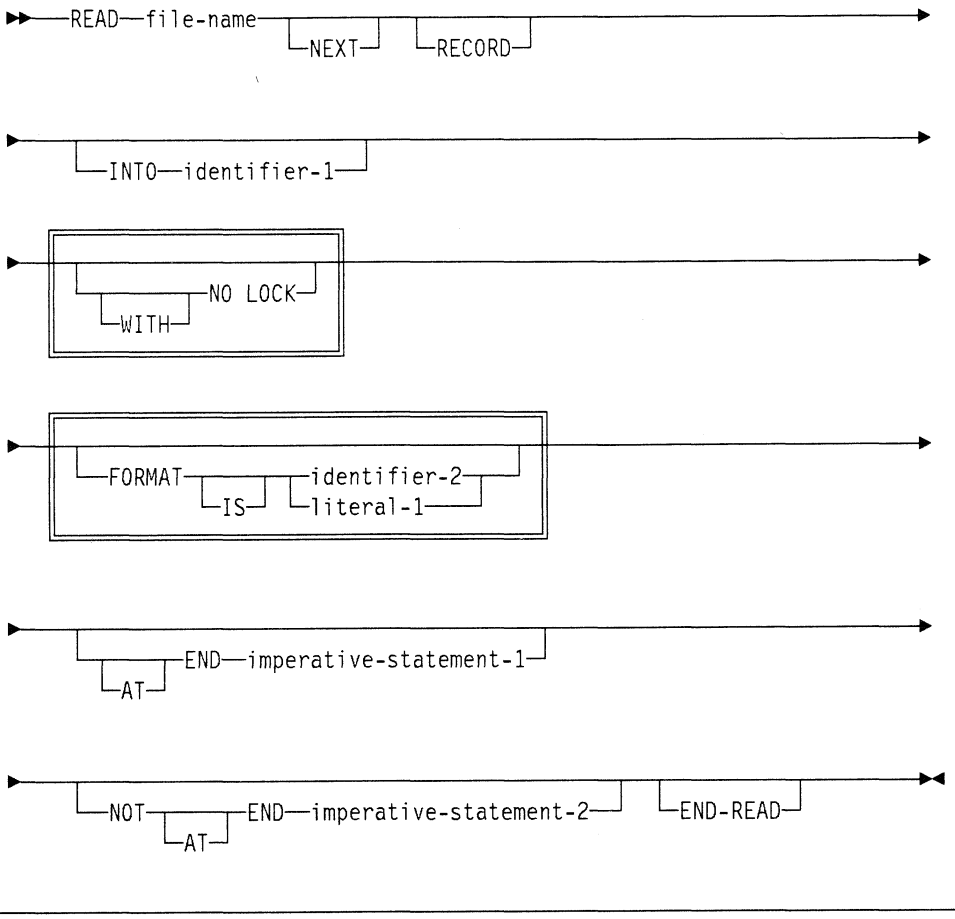
READ Statement

The READ statement makes a record available to the program.

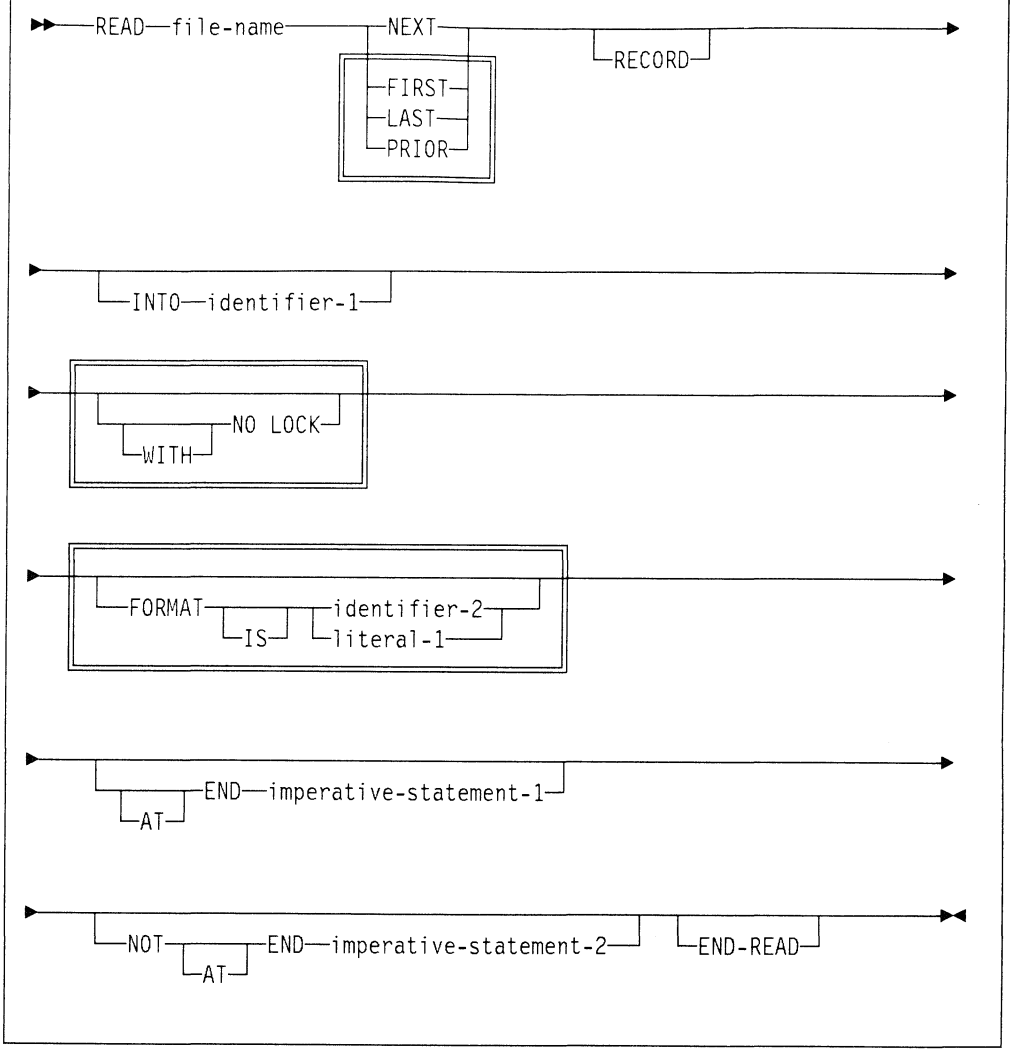
For sequential access, the READ statement makes the next logical record from a file available to the object program. For random access, the READ statement makes a specified record from a direct-access file available to the object program.

When the READ statement is executed, the associated file must be open in INPUT or I-O mode.

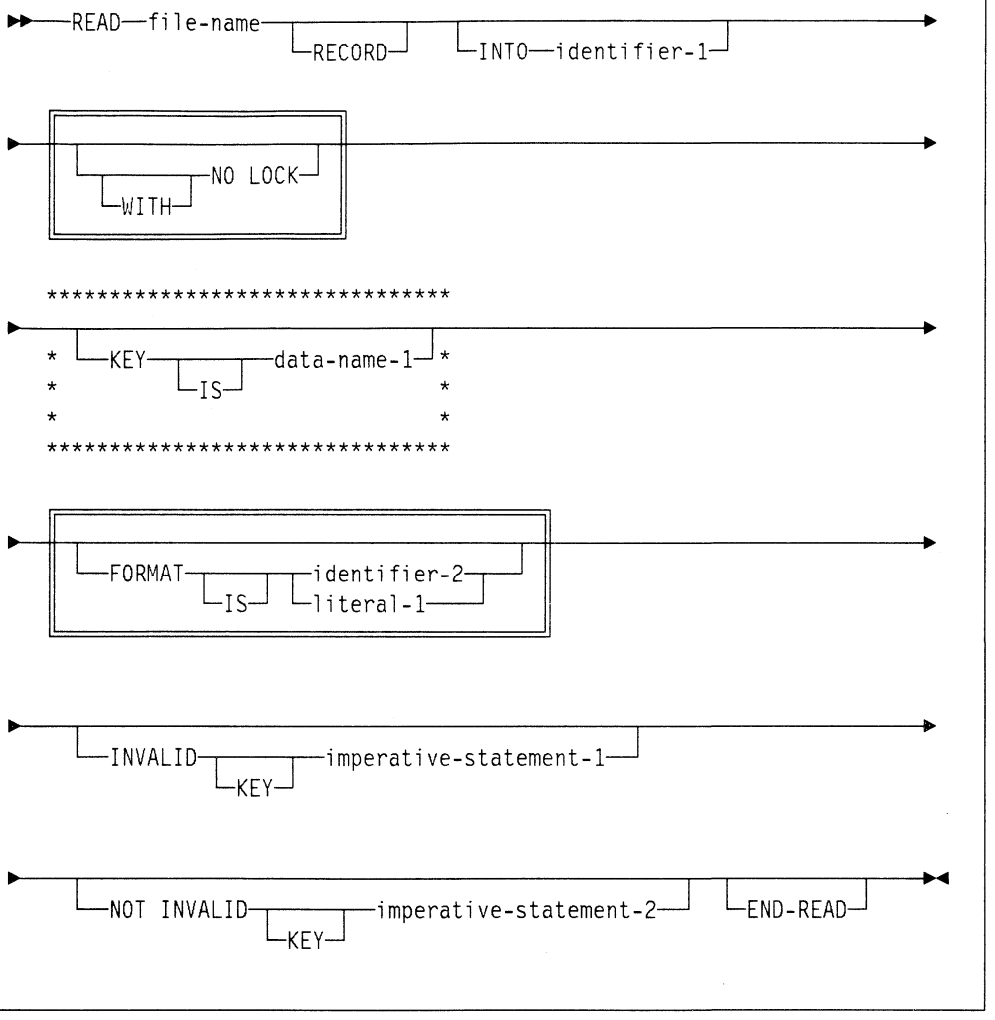
Format 1—Sequential Retrieval using SEQUENTIAL Access



Format 2—Sequential Retrieval using DYNAMIC Access



Format 3—Random Retrieval



file-name
File-name must be defined in a Data Division FD entry, and must not name a sort or merge file. If more than one record-description entry is associated with file-name, these records automatically share the same storage area. That is, they are implicitly redefined.

RECORD
The next record in the logical sequence of records.

KEY IS
This phrase is syntax checked only.

Sequential Access Mode

Format 1 must be used for all files in sequential access mode.

Execution of a Format 1 READ statement retrieves the next logical record from the file. The next record accessed is determined by the file organization.

Random Access Mode

Format 3 must be specified for indexed and relative files in random access mode, and also for files in the dynamic access mode when record retrieval is random.

Execution of the READ statement depends on the file organization, as explained in following sections.

Dynamic Access Mode

For files with indexed or relative organization, dynamic access mode may be specified in the FILE-CONTROL entry. In dynamic access mode, either sequential or random record retrieval can be used, depending on the format used.

Format 2 with the NEXT phrase must be specified for sequential retrieval. All other rules for sequential access apply.

Format 3 must be specified for random retrieval. All other rules for random access apply.

Multiple Record Processing

If more than one record description entry is associated with file-name-1, these records automatically share the same storage area; that is, they are implicitly redefined. After a READ statement is executed, only those data items within the range of the current record are replaced; data items stored beyond that range are undefined. Figure 24 illustrates this concept. If the range of the current record exceeds the record description entries for file-name, the record is truncated on the right to the maximum size. In either of these cases, the READ statement is successful and an I-O status is set indicating a record length conflict has occurred.

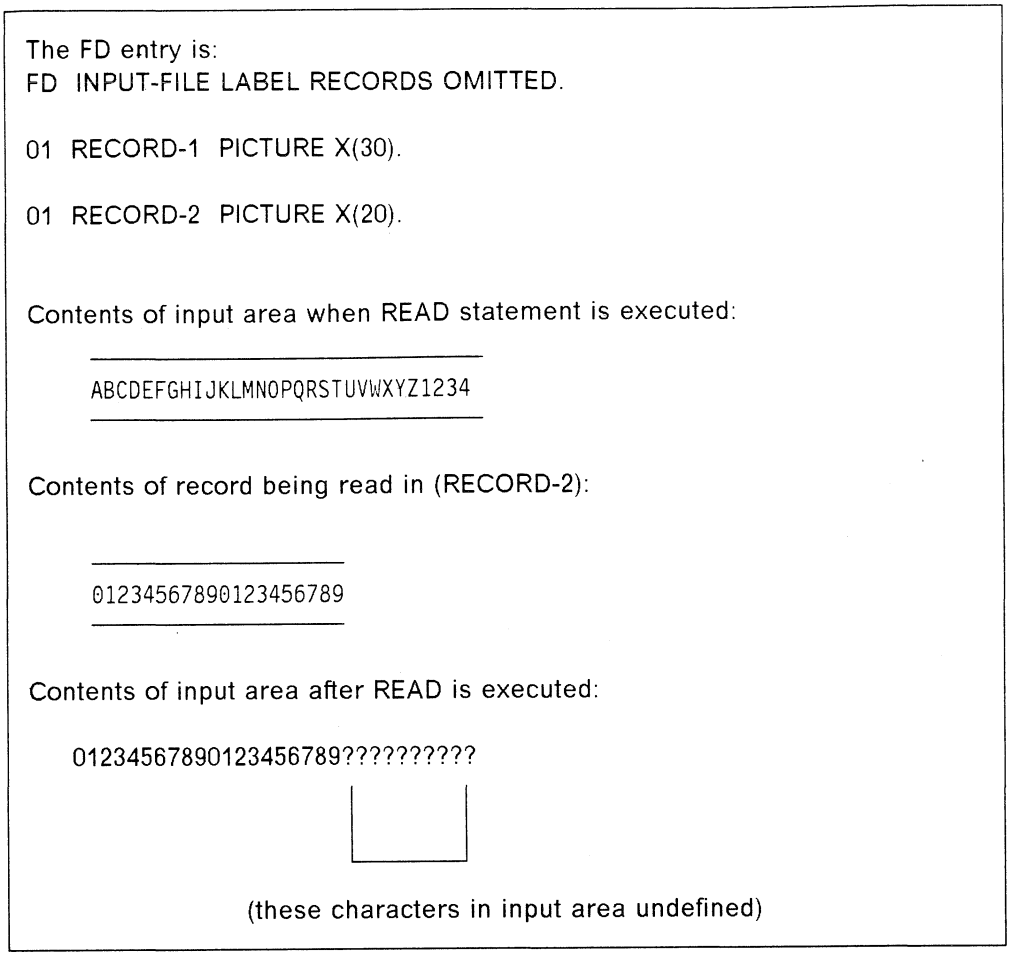


Figure 24. READ Statement with Multiple Record Description

Multivolume Files

If end-of-volume is recognized during execution of a READ statement, and logical end-of-file has not been reached, the following actions are taken:

- The system-defined ending volume label procedure
- A volume switch
- The system-defined beginning volume label procedure
- The first data record of the next volume is made available.

READ Statement Considerations

The following tables illustrate organization, access, and device considerations for the READ statement. The letter codes used in the tables are defined in the section following the tables.

Table 42. Sequential Organization

ACCESS	SEQUENTIAL					
DEVICE	**PRINTER**	**TAPEFILE**	**DISKETTE**	**DISK**	**DATABASE**	**FORMATFILE**
READ Verb	—	A,I,G1,S,V	A,I,G1,S,V	A,I,P,G1,S,N	A,I,P,G1,S,N	A,I,G1,S
NEXT	—	—	—	—	—	—
LAST	—	—	—	—	—	—
FIRST	—	—	—	—	—	—
PRIOR	—	—	—	—	—	—
INTO	—	O,B	O,B	O,B	O,B	O,B
NO LOCK	—	—	—	O,M	O,M	—
KEY IS	—	—	—	—	—	—
AT END	—	O,E,D1	O,E,D1	O,E,D1	O,E,D1	O,E,D1
NOT AT END	—	O,D3	O,D3	O,D3	O,D3	O,D3
INVALID KEY	—	—	—	—	—	—
NOT INVALID KEY	—	—	—	—	—	—
FORMAT	—	—	—	—	—	—

Table 43. Relative Organization

Device	DISK			DATABASE		
Access	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**
READ Verb	A,I,P,G2,N	A,I,P,G3,N	A,I,P,N	A,I,P,G2,N	A,I,P,G3,N	A,I,P,N
NEXT	—	—	O,Z1	—	—	O,Z1
FIRST	—	—	—	—	—	—
LAST	—	—	—	—	—	—
PRIOR	—	—	—	—	—	—
INTO	O,B	O,B	O,B	O,B	O,B	O,B
NO LOCK	O,M	O,M	O,M	O,M	O,M	O,M
KEY IS	—	—	—	—	—	—
AT END	O,E,D2	—	O,E,D2	O,E,D2	—	O,E,D2
NOT AT END	O,D3	—	O,D3	O,D3	—	O,D3
INVALID KEY	—	O,U1	O,U1	—	O,U1	O,U1
NOT INVALID KEY	—	O,U2	O,U2	—	O,U2	O,U2
FORMAT	—	—	—	—	—	—

Table 44. Indexed Organization

Device	DISK			DATABASE		
Access	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**
READ Verb	A,A2,I,P,G4,N	A,A2,I,P,G5,N	A,A2,I,P,N	A,A2,I,P,G4,N	A,A2,I,P,G5,N	A,A2,I,P,N
NEXT	—	—	O,Z2	—	—	O,Z3,Z4
FIRST	—	—	—	—	—	O,Z3
LAST	—	—	—	—	—	O,Z3
PRIOR	—	—	—	—	—	O,Z3,Z4
INTO	O,B	O,B	O,B	O,B	O,B	O,B
NO LOCK	O,M	O,M	O,M	O,M	O,M	O,M
KEY IS	—	—	—	—	—	—
AT END	O,E,D2	—	O,E,D2	O,E,D2	—	O,E,D2
NOT AT END	O,D3	—	O,D3	O,D3	—	O,D3
INVALID KEY	—	O,U1	O,U1	—	O,U1	O,U1
NOT INVALID KEY	—	O,U2	O,U2	—	O,U2	O,U2
FORMAT	—	—	—	O,F,X	O,F,W	O,F,Y

Letter Code **Meaning**

— Combination is not valid.

A If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the READ statement is processed.

Following the unsuccessful processing of any READ statement, the contents of the associated record area and the position of the file position indicator are undefined.

A2 To enable file status 02 for DUPLICATE KEY checking, you need:

- the WITH DUPLICATES phrase in the SELECT clause
- OPEN I-O or OPEN INPUT
- the *DUPKEYCHK option of the GENOPT parameter, or the DUPKEYCHK option of the PROCESS statement.

B The INTO identifier phrase makes a READ statement equivalent to:

```
READ file-name RECORD
MOVE record-name TO identifier
```

After successful processing of the READ statement, the current record becomes available both in the record-name and identifier.

When the INTO identifier phrase is specified, the current record is moved from the input area to the identifier area according to the rules for the MOVE statement without the CORRESPONDING phrase. Any subscripting, indexing, or reference modification associated with the identifier is evaluated after the record has been read and immediately before it is transferred to the identifier. (See also "INTO/FROM Identifier Phrase" on page 215.)

The INTO phrase may be specified in a READ if:

- Only one record description is subordinate to the file description entry,

or,

- All record-names associated with file-name, and the data item referenced by identifier-1, describe a group item, a numeric-edited item, or an elementary alphanumeric item.

The INTO identifier phrase cannot be specified when the file contains records of various sizes, as indicated by their record descriptions. The storage area associated with identifier and the record area associated with the file-name cannot be the same storage area.

D1 When the AT END condition is recognized, a successful CLOSE statement, followed by a successful OPEN statement, must be processed for this file before processing a READ statement.

D2 When the AT END condition is recognized, a sequential access READ statement for this file must not be processed without first processing one of the following:

- A successful CLOSE statement followed by a successful OPEN statement.
- A successful START statement for this file.
- A successful random access READ statement for this file.
- A successful READ file-name FIRST or READ file-name LAST where permitted.

D3 After each successful completion of a READ statement with the NOT AT END phrase (the high order digit of the file status is 0), control transfers to the imperative statement associated with the phrase.

E If no next logical record exists in the file when a sequential read is processed, an AT END condition occurs (the high order digit of the file status is 1), and READ statement processing is unsuccessful. The following actions are taken, in the order listed:

1. If the FILE STATUS clause is specified, the status key is updated to indicate an AT END condition.
2. If the AT END phrase is specified, control is transferred to the AT END imperative statement. Any EXCEPTION/ERROR procedure for this file is not run.
3. If the AT END phrase is not specified, any EXCEPTION/ERROR procedure for this file is run. Return from that procedure is to the next executable statement following the end of the READ statement.

   Note: A sequential read can be a read on a file with sequential access, or, a read NEXT, FIRST, LAST, or PRIOR, on a file with dynamic access. READ FIRST and READ LAST will only result in an AT END condition if there are no records in the file.

   When the AT END condition occurs, execution of the READ statement is unsuccessful. The contents of the associated record area are undefined and the file position indicator is set to indicate that no valid next record has been established.

   If a second sequential read is unsuccessful, a file status of 46 occurs and the AT END phrase is not executed.

   A READ PRIOR after an AT END condition will also result in a file status of 46. Neither the AT END phrase not the NOT AT END phrase will be executed.

If an AT END condition does not occur during the execution of a READ statement, the AT END phrase is ignored, if specified, and the following actions occur:

1. The file position indicator is set and the I-O status associated with file-name-1 is updated.
2. If an exception condition which is not an AT END condition exists, control is transferred according to rules of the USE statement following the execution of any USE AFTER EXCEPTION procedure applicable to file-name-1.
3. 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.

Following the unsuccessful execution of a READ statement, the contents of the associated record area are undefined and the file position indicator is set to indicate that no valid next record has been established.

The AT END phrase must be specified if no explicit or implicit EXCEPTION/ERROR procedure is specified for this file.

IBM Extension

F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format to operate on.

If an identifier is specified, it must be a character-string of ten characters or less, and it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

If a literal is specified, it must be an uppercase character-string of ten characters or less.

A value of all blanks is treated as though the FORMAT phrase were not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is invoked, if applicable for the file.

End of IBM Extension

G1 The record that is made available by the READ statement is determined as follows:

- If the file position indicator was set by the processing of an OPEN statement, the record pointed to is made available.

- If the file position indicator was set by the processing of a previous READ statement, the pointer is updated to point to the next existing record in the file. That record is then made available.

G2 The record that is made available by the READ statement is determined as follows:

- If the file position indicator was set by the processing of a START or OPEN statement, the record pointed to is made available if it is still accessible through the path indicated by the file position indicator. If the record is no longer accessible (due, for example, to deletion of the record), the current record pointer is updated to indicate the next existing record in the file. That record is then made available.
- If the file position indicator was set by the processing of a previous READ statement, the file position indicator is updated to point to the next existing record in the file. That record is then made available.

If the RELATIVE KEY phrase is specified for this file, READ statement processing updates the RELATIVE KEY data item to indicate the relative record number of the record being made available.

G3 The record with the relative record number contained in the RELATIVE KEY data item is made available. If the file does not contain such a record, the INVALID KEY condition exists, and READ statement processing is unsuccessful.

G4 The record made available by the READ statement is determined as follows:

- If the file position indicator was set by the processing of a START or OPEN statement, the record pointed to is made available if it is still accessible through the path indicated by the current record pointer. If the record is no longer accessible (due, for example, to deletion of the record), the file position indicator is updated to indicate the next existing record in the file. That record is then made available.
- If the file position indicator was set by the processing of a previous READ statement, the file position indicator is updated to point to the next existing record in the file. That record is then made available.

IBM Extension

For a file that allows duplicate keys (the DUPLICATES phrase is specified in the file-control entry), the records with duplicate key values are made available in the order specified when the file was created. The system options are first-in first-out (FIFO), last-in first-out (LIFO), and no specific sequence (if neither LIFO not FIFO is specified).

End of IBM Extension

G5 The record in the file with a key value equal to that of the RECORD KEY data item is then made available. If the file does not contain such a record, the INVALID KEY condition exists, and READ statement processing is unsuccessful.

IBM Extension

For a file that allows duplicate keys (the DUPLICATES phrase is specified in the file-control entry), the first record with the specified key value is made available. The first record is determined by the order specified when the file was created. The system options are first-in first-out (FIFO), last-in first-out (LIFO), and no specific sequence (if neither LIFO not FIFO is specified).

End of IBM Extension

I Allowed when the file is opened for INPUT.

IBM Extension

M The NO LOCK phrase prevents the READ operation from obtaining record locks on files that you open in I-O (update) mode. A READ statement bearing this phrase also releases records that have been locked by a previous READ operation.

If you use this phrase for a file that is not open in I-O mode, you receive an error message.

End of IBM Extension

For information about file and record locking, see the *COBOL/400 User's Guide*.

N Null-capable fields are supported, but null values are not. Null values result in a file status of 90.

O You can specify this combination.

P Allowed when the file is opened for I-O.

S If SELECT OPTIONAL is specified in the file-control entry for this file and the file is not available when this program runs, processing of the first READ statement causes an AT END condition. Since the file is not available, the standard system end-of-file processing is not done when the file is closed.

U1 The INVALID KEY phrase must be specified for files for which there is not an appropriate EXCEPTION/ERROR procedure.

For information about INVALID KEY phrase processing, see "INVALID KEY Condition" on page 215.

U2 After the successful completion of a READ statement with the NOT INVALID KEY phrase, control transfers to the imperative statement associated with the phrase.

V If end of volume is recognized during processing of a READ statement and logical end of file has not been reached, the following actions are taken in the order listed:

1. The standard ending volume label procedure is processed.
2. A volume switch occurs.
3. The standard beginning volume label procedure is run.
4. The first data record of the next volume is made available.

The program receives no indication that the above actions occurred during the read operation.

W If specified, the key as defined for the specified format is used to get a record of that format. If a record of that format is not found, a record-not-found condition is returned. This occurs even when there are records that have the defined key, but that have a different record format.

If the format is omitted, the common key for the file is used to get the first record of any format that has that common key value. The common key for a file consists of the key fields common to all formats of a file for records residing on the database. The common key for a file is the left-most key fields that are common across all record formats in the file. The common key is built from the data in the record description area using the first record format defined in the program for the file.

X If specified, the next record in the keyed sequence access path that has the requested format is made available. If omitted, the next record in the keyed sequence access path is made available.

Y

	FORMAT Phrase	
	Specified	Omitted
NEXT	Y1	Y2
PRIOR	Y3	Y4
FIRST	Y5	Y6
LAST	Y7	Y8
None of the above	Y9	Y10

Y1 The next record in the keyed sequence access path having the specified format is made available.

Y2 The next record in the keyed sequence access path is made available regardless of its format.

Y3 The record in the keyed sequence access path preceding the record identified by the file position indicator having the specified format is made available.

Y4 The record in the keyed sequence access path preceding the record identified by the file position indicator is made available regardless of its format.

Y5 The first record in the keyed sequence access path having the specified format is made available.

Y6 The first record in the keyed sequence access path is made available regardless of its format.

Y7 The last record in the keyed sequence access path having the specified format is made available.

Y8 The last record in the keyed sequence access path is made available regardless of its format.

Y9 The key as defined for the specified format is used to get a record of that format. If a record of that format is not found, a record-not-found condition is returned. This occurs even when there are records that have the defined key, but that have a different record format.

Y10 The common key for the file is used to get the first record of any format that has that common key value. The common key for a file consists of the key fields common to all formats of a file for records residing on the database. The common key for a file consists of the leftmost key fields that are common across all record formats in the file. The common key is built from the data in the record description area using the first record format defined in the program for the file.

Z1 When specified, a sequential read is done (see G2). When omitted, a random read is done (see G3).

Z2 When specified, a sequential read is done (see G4). When omitted, a random access read is done (see G5).

Z3 When specified, a sequential read is done (see G4). If NEXT, FIRST, LAST and PRIOR are all omitted, a random access read is done (see G5).

Z4 If you perform a READ NEXT operation on a block of records, you cannot perform a READ PRIOR operation until the block is empty. The converse is also true.

To recover from file status 9U, close the file, then open it again.

READ Statement Notes

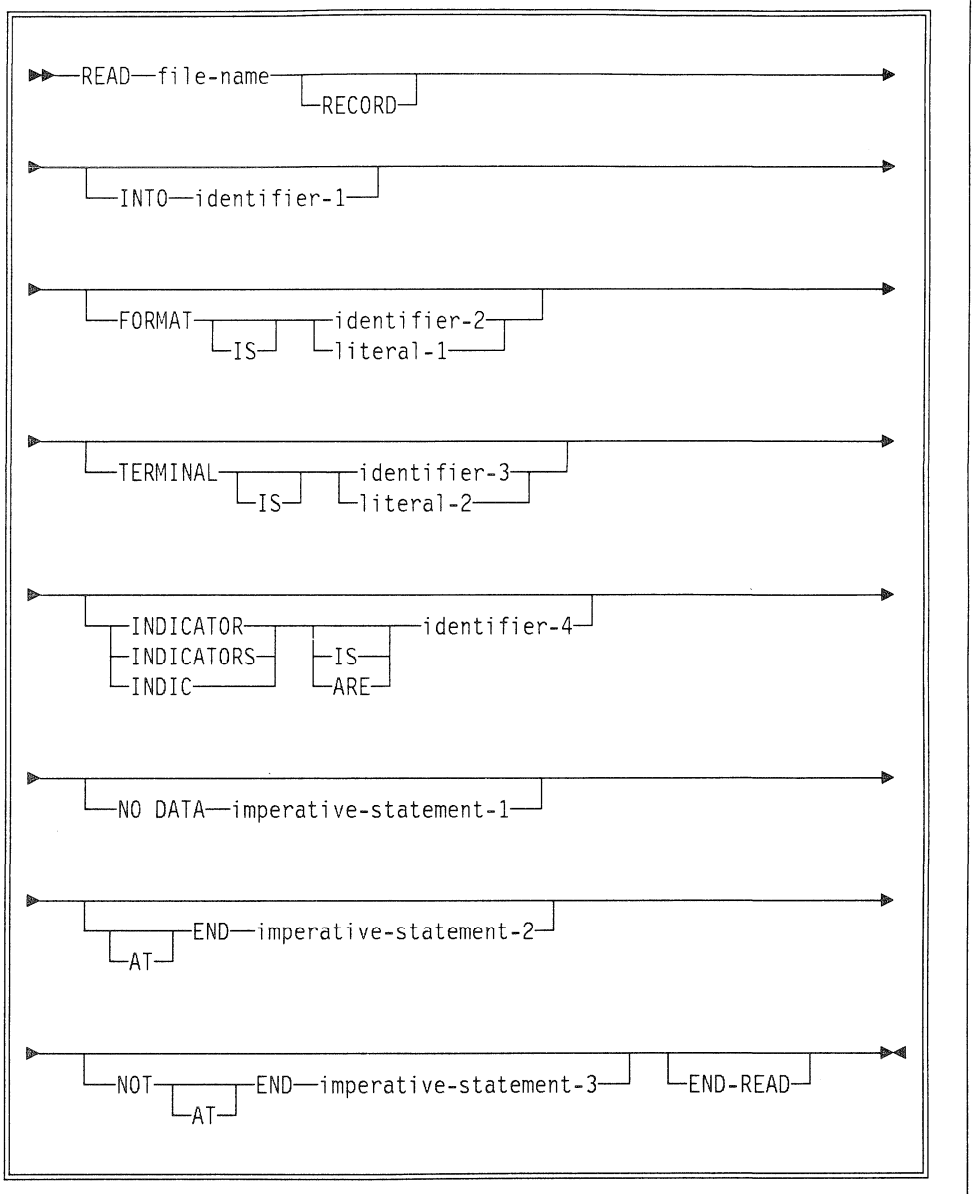
If the FILE STATUS clause is specified in the FILE-CONTROL entry, the associated status key is updated when the READ statement is executed.

Following unsuccessful READ statement execution, the contents of the associated record area and the value of the file position indicator are undefined.

Transaction Files

The READ statement makes a record from a device available, using a named format. If the format is a subfile, the READ statement makes a specified record available from that subfile.

Format 4—TRANSACTION (Nonsubfile)



Format 4 is used only to read a format that is not a subfile record. The RELATIVE KEY data item, if specified in the FILE-CONTROL entry, is not used. The Format 4 READ statement is not valid for a subfile record; however, a Format 4 READ statement for the subfile control record format must be used to put those subfile records that were updated on a display into the subfile.

If the data is available, it is returned in the record area. The names of the record format and the program device are returned in the I-O-FEEDBACK area in the CONTROL-AREA.

The READ statement is valid only when there are acquired devices for the file. If a READ is executed and there are no acquired devices, the file status is set to 92 (logic error).

The manner in which the Format 4 READ statement functions depends on:

- If the READ is for a single device file or a multiple device file
- If a specific program device has been requested through the TERMINAL phrase
- If a specific record format has been requested through the FORMAT phrase
- If the NO DATA phrase has been specified.

In the following discussions, references to "data available or returned" include the situation where only the response indicators are set. This is so even when a separate indicator area is used and the indicators are not returned in the record area for the file.

The following chart shows the possible combinations of phrases, and the function performed for a single device file or a multiple device file. For example, if TERMINAL is N, FORMAT is N, and NO DATA is N, then the single device is D and multiple device is A.

	Phrase	Y=Yes N=No
Checked at Compilation	TERMINAL[2] FORMAT[2] NO DATA	N N N N Y Y Y Y N N Y Y N N Y Y N Y N Y N Y N Y
Determined at Execution	Single Device Multiple Device	D C D B D C D B A A D B D C D B

Codes A through D are explained below.

Code A—Read From Invited Program Device (Multiple Device Files only)

This type of READ receives data from the first invited program device that has data available. An invited program device is a work station or communications device (such as APPC, SNUF, BSCEL, Asynchronous Communications) that has been invited to send input. Inviting is done by writing to the program device with a format that has the DDS keyword INVITE specified. Once an invited program device is actually read from, it is no longer invited. That program device will not be used for input by another READ statement unless reinvited, or unless a READ is directed to it specifying the TERMINAL phrase or FORMAT phrase.

The record format returned from the program device is determined by the system. See the chapter on display devices in the *Data Management Guide* for information on how this is determined for work stations. For communications

[2] If the phrase is specified and the data item or literal is blank, the phrase is treated at execution time as if it were not specified.

devices, see the *ICF Programmer's Guide* for more information on format selection processing for an ICF file.

This READ can complete without returning any data in the following cases:

1. There are no invited devices and the timer function is not in effect. (This is the AT END condition.)
2. A controlled cancellation of the job occurs. This results in a file status value of 9A and a major-minor return code value of 0309.
3. The NO DATA phrase is omitted and the specified wait time expires. This results in a file status value of 00 and a major-minor return code value of 0310. The specified wait time is the value entered on the WAITRCD parameter for the file or the time interval specified on the timer function.
4. The NO DATA phrase is specified and there is no data immediately available when the READ is executed.

If data is available, it is returned in the record area. The record format is returned in the I-O-FEEDBACK area and in the CONTROL-AREA. For more information about "reading from invited program devices," see the *Data Management Guide* for display stations, and the *ICF Programmer's Guide* for communications devices.

Code B—Read From One Program Device (Invalid combination)

A compilation time message is issued and the NO DATA phrase is ignored. See the table entry for the same combination of phrases with the NO DATA phrase omitted.

Code C—Read From One Program Device (with NO DATA phrase)

This function of the READ statement never causes program execution to stop and wait until data is available. Either the data is immediately available or the NO DATA imperative statement is executed.

This READ function can be used to periodically check if data is available from a particular program device (either the default program device or one specified by the TERMINAL phrase). This checking for data is done in the following manner:

1. The program device is determined as follows:
   a. If the TERMINAL phrase was omitted or contains blanks, the default program device is used. The default program device is the one used by the last attempted READ, WRITE, REWRITE, ACQUIRE, or DROP statement. If none of the above I-O operations were previously executed, the default program device is the first program device acquired.
   b. If the TERMINAL phrase was specified, the indicated program device is used.
2. A check is done to determine if data is available and if the program device is invited.
3. If data is available, that data is returned in the record area and the program device is no longer invited. If no data is immediately available, the NO DATA imperative statement is executed and the program device remains invited.
4. If the program device is not invited, the AT END condition exists and the file status is set to 10.

Code D – Read From One Program Device (without NO DATA Phrase)

This READ always waits for data to be made available. Even if the job receives a controlled cancellation, or a WAITRCD time is specified for the file, the program will never regain control from the READ statement. This READ operation is performed in the following manner:

1. The program device is determined as follows:
   a. If the TERMINAL phrase is omitted or contains a blank value, the default program device is used. The default program device is the program device used by the last attempted READ, WRITE, REWRITE, ACQUIRE, DROP or ACCEPT (Attribute Data) statement. If none of these operations has been done, the program device implicitly acquired when the file was opened is used. If there are no acquired devices, the AT END condition exists.
   b. If the TERMINAL phrase is specified, the indicated program device is used.
2. The record format is determined as follows:
   a. If the FORMAT phrase is omitted or contains blanks, the record format returned is determined by the system. For information on how the record format is determined for work station devices, refer to the *Data Management Guide*. For information about how the record format is determined for communications devices, see the section on the FMTSLT parameter for the ADDICFDEVE and OVRICFDEVE commands in the *ICF Programmer's Guide*.
   b. If the FORMAT phrase is specified, the indicated record format is returned. If the data available does not match the requested record format, a file status of 9K is set.
3. Program execution stops until data becomes available. The data is returned in the record area after the READ statement is executed. If the program device was previously invited, it will no longer be invited after this READ statement.

INTO Phrase

The INTO phrase cannot be specified unless:

- all records associated with the file and the data item specified in the INTO phrase are group items, numeric-edited items, or elementary alphanumeric items.

OR

- only one record description is subordinate to the file description entry.

KEY IS Phrase

The KEY IS phrase may be specified only for indexed files. Data-name must identify a record key associated with file-name-1. Data-name-1 may be qualified; it may not be subscripted.

Note: The KEY IS phrase is syntax checked only and has no effect on the operation of the READ statement.

AT END Phrase

Imperative-statement-2 is executed when the AT END condition is detected.

FORMAT Phrase

Literal-1 or identifier-2 specifies the name of the record format to be read. Literal-1, if specified, must be nonnumeric, uppercase, and 10 characters or less in length. Identifier-2, if specified, must refer to an alphanumeric data item, 10 characters or less in length. If identifier-2 contains blanks, the READ statement is executed as if the FORMAT phrase were omitted.

NO DATA Phrase

When the NO DATA phrase is specified, the READ statement will determine whether data is immediately available. If data is available, the data is returned in the record area. If no data is immediately available, imperative-statement-1 is executed. The NO DATA phrase prevents the READ statement from waiting for data to become available.

TERMINAL Phrase

Literal-2 or identifier-3 specifies the program device name. Literal-2, if specified, must be nonnumeric and 10 characters or less in length. Identifier-3, if specified, must refer to an alphanumeric data item, 10 characters or less in length. The program device must have been acquired before the READ statement is executed. If identifier-3 contains blanks, the READ statement is executed as if the TERMINAL phrase was omitted. For a single device file, the TERMINAL phrase can be omitted. The program device is assumed to be that single device.

If the TERMINAL phrase is omitted for a READ of a TRANSACTION file that has acquired multiple program devices, the default program device is used.

INVALID KEY Phrase

If it is time to run the statement, and the RELATIVE KEY data item contains a value that does not correspond to a relative record number for the subfile, the INVALID KEY condition exists and the running of the statement is unsuccessful.

For a Format 2 READ, the INVALID KEY phrase should be specified if the NEXT MODIFIED phrase is not specified and there is no applicable USE procedure specified for the file-name.

For information about what happens when the invalid key condition occurs, see "INVALID KEY Condition" on page 215.

NOT INVALID KEY Phrase

This phrase allows you to specify procedures that will be performed when an invalid key condition does not exist for the statement that is used.

AT END Phrase

The AT END phrase serves to explicitly delimit the scope of the statement. Imperative-statement-2 is executed when the AT END condition is detected.

If the NEXT MODIFIED phrase is specified and there is no user-modified record in the subfile, the AT END condition exists, and the running of the statement is unsuccessful.

The AT END phrase should be specified when the NEXT MODIFIED phrase is used, and no applicable USE procedure is specified for the file-name. If the AT END phrase and a USE procedure are both specified for a file, and the AT END condition arises, control transfers to the AT END imperative statement and the USE procedure is not run.

NOT AT END Phrase

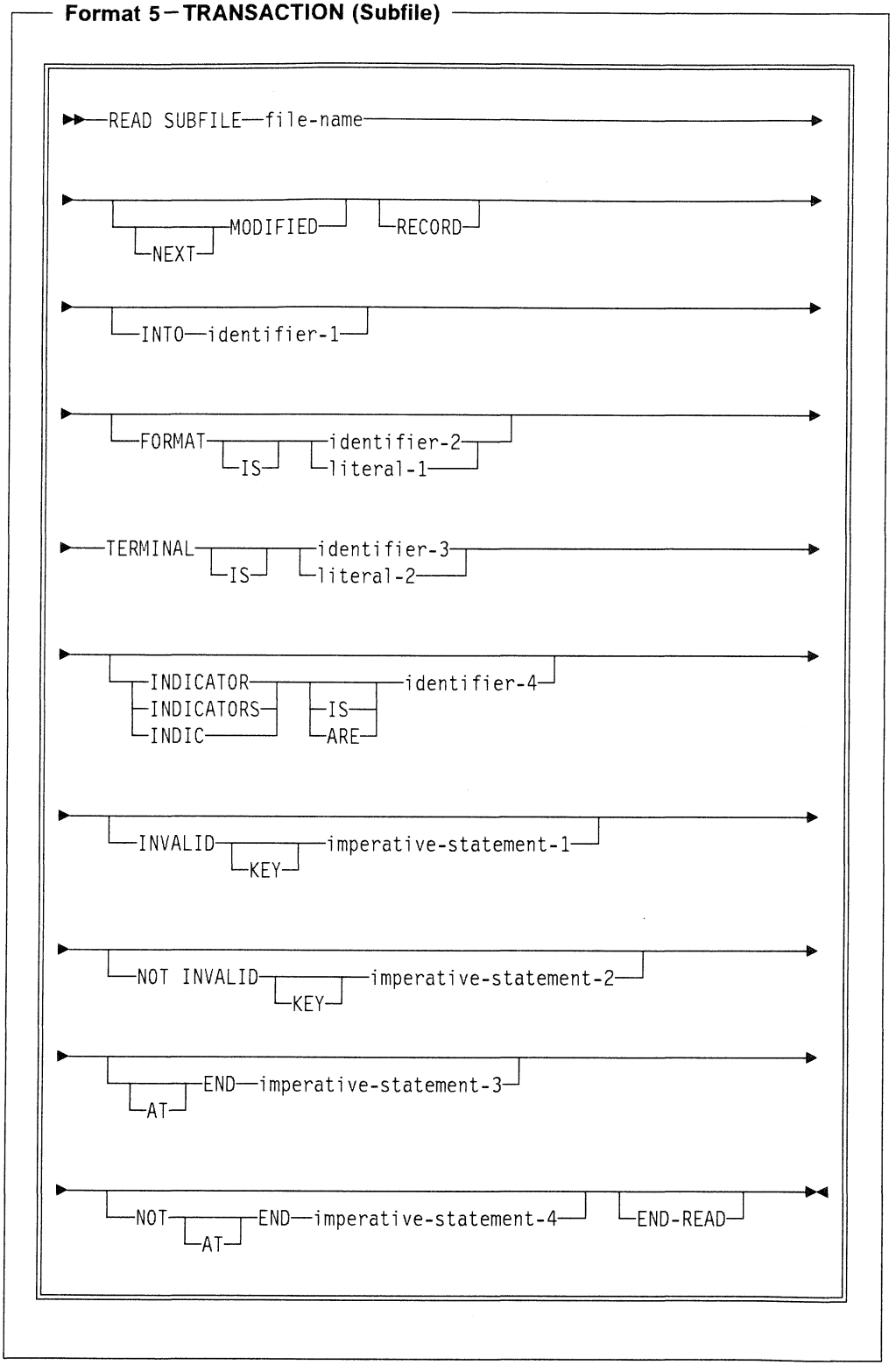
This phrase allows you to specify procedures that will be performed when the AT END condition does not exist for the statement that is used.

END-READ Phrase

This explicit scope terminator serves to delimit the scope of the READ statement. END-READ permits a conditional READ statement to be nested in another conditional statement. END-READ may also be used with an imperative READ statement.

For more information, see "Delimited Scope Statements" on page 207.

Format 5—TRANSACTION (Subfile)



Format 5 is used only to read a format that is a subfile record. The AT END phrase can only be used when the NEXT MODIFIED phrase is specified. The INVALID KEY phrase must not be used when the NEXT MODIFIED phrase is specified.

Format 5 cannot be used for communications devices. If the subfile format of the READ statement is used for a communications device, the READ fails and a file status of 90 is set.

Random Access of Subfile Records: The NEXT MODIFIED phrase must not be used to randomly access records in a subfile. The INVALID KEY phrase can only be used for random access of subfile records.

Sequential Access of Subfile Records: The NEXT MODIFIED phrase must be specified to access subfile records sequentially. The AT END phrase can only be specified with the NEXT MODIFIED phrase.

NEXT MODIFIED Phrase

When NEXT MODIFIED is not specified, the data record made available is the record in the subfile with a relative record number that corresponds to the value of the RELATIVE KEY data item.

When the NEXT MODIFIED phrase is not specified, and if the RELATIVE KEY data item contains a value other than the relative record number of a record in the subfile, the INVALID KEY condition exists and the execution of the READ statement is unsuccessful.

When the NEXT MODIFIED phrase is specified, the record made available is the first record in the subfile that has been modified (has the Modified Data Tag on). For information about turning on the Modified Data Tag, see the *Data Management Guide*.

The search for the next modified record begins:

- At the beginning of the subfile if:
  - An I-O operation has been performed for the subfile control record.
  - The I-O operation cleared, initialized, or displayed the subfile.
- For all other cases, with the record following the record that was read by a previous read operation.

The value of the RELATIVE KEY data item is updated to reflect the relative record number of the record made available to the program.

If NEXT MODIFIED is specified and there is no user-modified record in the subfile with a relative record number greater than the relative record number contained in the RELATIVE KEY data item, the AT END condition exists, the file status is set to 12, and the value of the RELATIVE KEY data item is set to the key of the last record in the subfile. Imperative-statement-2, or any applicable USE AFTER ERROR/EXCEPTION procedure, if any, is then executed.

FORMAT Phrase

When a format-name is not specified, the format used is the last record format written to the display device that contains input fields, input/output fields, or hidden fields. If no such format exists for the display file, the format used is the record format of the last WRITE operation to the display device.

If the FORMAT phrase is specified, literal-1 or the contents of identifier-2 must specify a format, which is active for the appropriate program device. The READ statement reads a data record of the specified format.

The FORMAT phrase should always be specified for multiple format files to ensure correct results.

TERMINAL Phrase

See Format 4 above for general considerations concerning the TERMINAL phrase.

For a Format 5 READ, if the TERMINAL phrase is omitted for a file that has multiple devices acquired for it, a record is read from the subfile associated with the default program device.

INVALID KEY Phrase

If the RELATIVE KEY data item at the time of the execution of the READ statement contains a value that does not correspond to a relative record number for the subfile, the INVALID KEY condition exists and the execution of the READ statement is unsuccessful.

For a Format 5 READ, the INVALID KEY phrase should be specified if the NEXT MODIFIED phrase is not specified and there is no applicable USE procedure specified for the file-name.

For information about what happens when the invalid key condition occurs, see "INVALID KEY Condition" on page 215.

NOT INVALID KEY Phrase

This phrase allows you to specify procedures that will be performed when an invalid key condition does not exist for the statement that is used.

AT END Phrase

If NEXT MODIFIED is specified and there is no user-modified record in the subfile, the AT END condition exists, and the execution of the READ statement is unsuccessful.

The AT END phrase should be specified when the NEXT MODIFIED phrase is used, and no applicable USE procedure is specified for the file-name. If the AT END phrase and a USE procedure are both specified for a file, and the AT END condition arises, control transfers to the AT END imperative statement and the USE procedure is not executed.

NOT AT END Phrase

This phrase allows you to specify procedures that will be performed when the AT END condition does not exist for the statement that is used.

END-READ Phrase

This explicit scope terminator serves to delimit the scope of the READ statement. END-READ permits a conditional READ statement to be nested in another conditional statement. END-READ may also be used with an imperative READ statement.

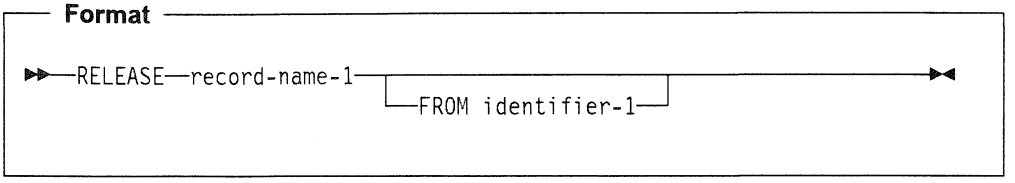
For more information, see "Delimited Scope Statements" on page 207.

RELEASE Statement

The RELEASE statement transfers records from an input/output area to the initial phase of a sorting operation.

The RELEASE statement can only be used within the range of an input procedure associated with a SORT statement.

Format



Within an INPUT PROCEDURE, at least one RELEASE statement must be specified.

When the RELEASE statement is executed, the current contents of record-name-1 are placed in the sort file; that is, made available to the initial phase of the sorting operation.

record-name-1
Must specify the name of a logical record in a sort-merge file description entry (SD). Record-name-1 may be qualified.

FROM identifier-1
Makes the RELEASE statement equivalent to the statements:

```
MOVE identifier-1 to record-name-1
RELEASE record-name-1
```

Moving takes place according to the rules for the MOVE statement without the CORRESPONDING phrase.

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

If the RELEASE statement is executed without specifying the SD entry for file-name-1 in a SAME RECORD AREA clause, the information in record-name-1 is no longer available.

If the SD entry **is** specified in a SAME RECORD AREA clause, record-name-1 is still available as a record of the other files named in that clause.

When FROM identifier-1 is specified, the information is still available in identifier-1.

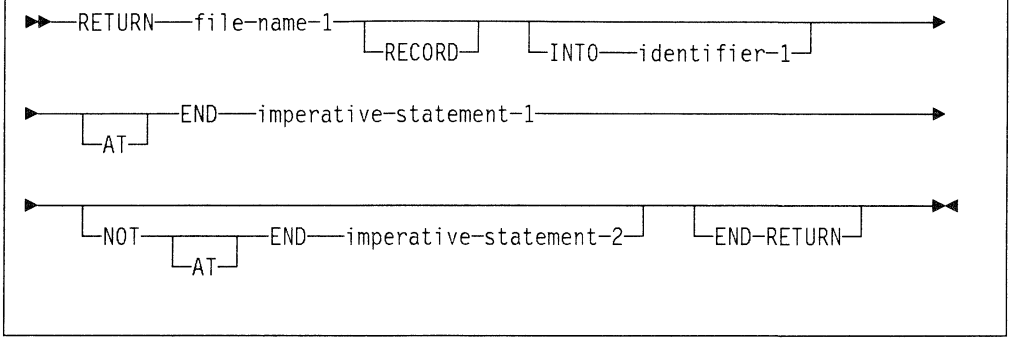
When control passes from the INPUT PROCEDURE, the sort file consists of all those records placed in it by execution of RELEASE statements.

RETURN Statement

The RETURN statement transfers records from the final phase of a sort or merge operation to an OUTPUT PROCEDURE.

The RETURN statement can be used only within the range of an output procedure associated with a SORT or MERGE statement.

Format



Within an OUTPUT PROCEDURE, at least one RETURN statement must be specified.

When the RETURN statement is executed, the next record from file-name-1. is made available for processing by the OUTPUT PROCEDURE.

file-name-1

Must be described in a Data Division SD entry.

If more than one record description is associated with file-name-1, these records automatically share the same storage; that is, the area is implicitly redefined. After RETURN statement execution, only the contents of the current record are available; if any data items lie beyond the length of the current record, their contents are undefined.

INTO identifier-1

The RETURN INTO statement is equivalent to the statements:

```
RETURN file-name-1
MOVE record-name TO identifier-1
```

Moving takes place according to the rules for the MOVE statement without the CORRESPONDING phrase. Any subscripting, indexing, or reference modification associated with identifier-1 is evaluated after the record has been returned and immediately before it is moved to identifier-1.

The INTO phrase may be specified in a RETURN statement if one or both of the following are true:

- If only one record description is subordinate to the sort-merge file description entry, and/or
- If all record-names associated with file-name-1 and the data item referenced by identifier-1 describe a group item, a numeric-edited item, or an elementary alphanumeric item.

The record areas associated with file-name-1 and identifier-1 must not be the same storage area.

AT END Phrases

The imperative-statement specified on the AT END phrase executes after all records have been returned from file-name-1. No more RETURN statements may be executed as part of the current output procedure.

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 the imperative statement specified by the NOT AT END phrase, otherwise control is passed to the end of the RETURN statement.

END-RETURN Phrase

This explicit scope terminator serves to delimit the scope of the RETURN statement. END-RETURN permits a conditional RETURN statement to be nested in another conditional statement. END-RETURN may also be used with an imperative RETURN statement.

For more information, see "Delimited Scope Statements" on page 207.

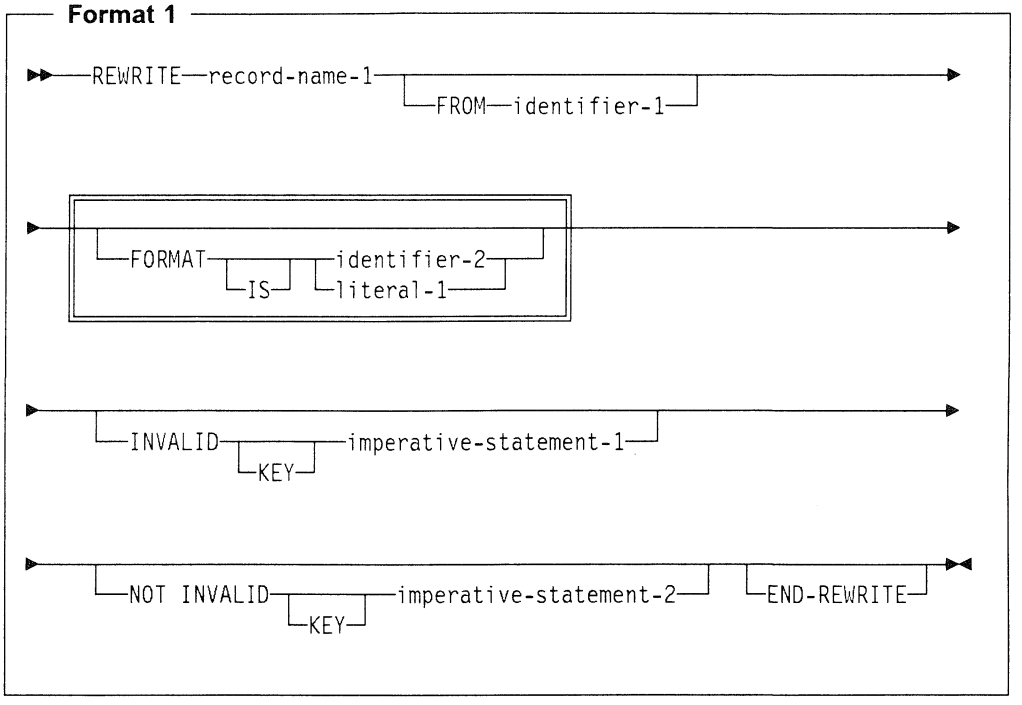
REWRITE Statement

The REWRITE statement logically replaces an existing record in a direct-access file.

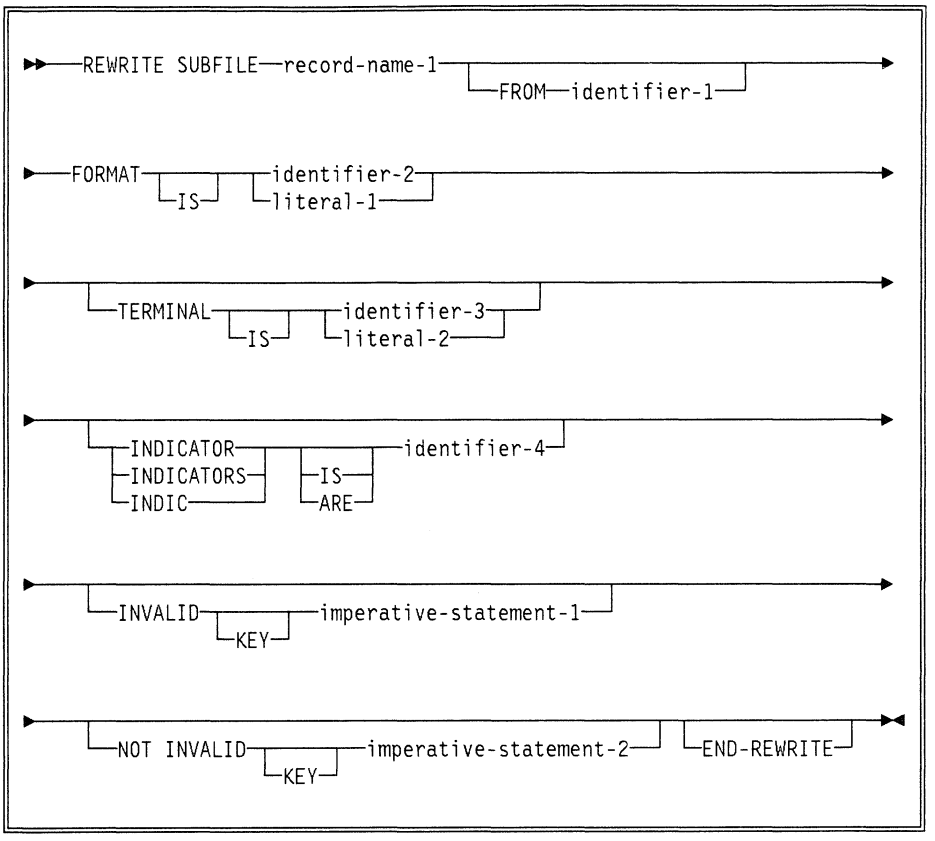
When the REWRITE statement is executed, the associated direct-access file must be open in I-O mode.

The formats for the REWRITE statement are as follows:

Format 1



Format 2—TRANSACTION (Subfile)



record-name-1

- Must be the name of a record in the File Section
- Must have the same number of character positions as the record being replaced
- Must not be subscripted, indexed, or reference modified
- Can be qualified.

FROM identifier-1

After successful REWRITE statement execution, the logical record is still available in **identifier-1**.

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

Reusing a Logical Record

After successful execution of a REWRITE statement, the logical record is no longer available in record-name-1 unless the associated file is named in a SAME RECORD AREA clause (in which case, the record is also available as a record of the other files named in the SAME RECORD AREA clause).

The file position indicator is not affected by execution of the REWRITE statement.

If the FILE STATUS clause is specified in the FILE-CONTROL entry, the associated status key is updated when the REWRITE statement is executed.

REWRITE Statement Considerations

The following tables illustrate organization, access, and device considerations for the REWRITE statement. The letter codes used in the tables are defined in the section following the tables.

Table 45. Sequential Organization

ACCESS	SEQUENTIAL					
DEVICE	**PRINTER**	**TAPEFILE**	**DISKETTE**	**DISK**	**DATABASE**	**FORMATFILE**
REWRITE Verb	—	—	—	A,P	A,P	—
FROM	—	—	—	O,B	O,B	—
INVALID KEY	—	—	—	—	—	—
NOT INVALID KEY	—	—	—	—	—	—
FORMAT	—	—	—	—	—	—

Table 46. Relative Organization

Device	DISK			DATABASE		
Access	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**
REWRITE Verb	A,P,Z	S,P,Z	S,P,Z	A,P,Z	S,P,Z	S,P,Z
FROM	O,B	O,B	O,B	O,B	O,B	O,B
INVALID KEY	—	O,U1,H,J	O,U1,H,J	—	O,U1,H,J	O,U1,H,J
NOT INVALID KEY	—	O,U2	O,U2	—	O,U2	O,U2
FORMAT	—	—	—	—	—	—

Table 47. Indexed Organization

Device	DISK			DATABASE		
Access	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**
REWRITE Verb	A,E,P,Z	D,P,Z	D,P,Z	A,E,P,Z	D,P,Z	D,P,Z
FROM	O,B	O,B	O,B	O,B	O,B	O,B
INVALID KEY	O,U1,G,J	O,U1,H,J	O,U1,H,J	O,U1,H,J	O,U1,H,J	O,U1,H,J
NOT INVALID KEY	O,U2	O,U2	O,U2	O,U2	O,U2	O,U2
FORMAT	—	—	—	—	M,F	M,F

Letter Code	Meaning
—	Combination is not valid.
A	When the REWRITE statement is processed, the system logically replaces a record retrieved by a READ statement. The last input/output statement for this file must have been a successfully processed READ statement **without** the NO LOCK phrase.

IBM Extension

If the last input/output statement was a successfully processed READ statement **with** the NO LOCK phrase:

- The file status key, if defined, is set to 9S.
- The EXCEPTION/ERROR procedure, if any, is run.
- The REWRITE statement is not processed.

End of IBM Extension

If the last input/output statement was not a successfully processed READ statement, the file status key (if defined) is set to 43.

See the *COBOL/400 User's Guide* for information about file and record locking.

B The FROM identifier phrase makes a REWRITE statement equivalent to:

```
MOVE identifier TO record-name
REWRITE record-name.
```

After successful processing of the REWRITE statement, the current record is no longer available in record-name, but is still available in identifier. Record-name and identifier cannot both refer to the same storage area.

D The record to be replaced is specified by the value in the RECORD KEY data item. If the file does not contain such a record, an INVALID KEY condition exists.

IBM Extension

When EXTERNALLY-DESCRIBED-KEY is specified for the file, the key field in the record area for the format specified by the FORMAT phrase (or, if not specified, the first format defined in the program for the file), is used to find the record to be replaced.

DUPLICATES Phrase

If this phrase was specified for the file, the last input/output statement processed for this file before the REWRITE statement must have been a successfully processed READ statement **without** the NO LOCK phrase. The record read by that statement is the one that is replaced. In this case, the FORMAT phrase is not used in determining the record to be replaced.

The READ statement is required to ensure that the proper record is replaced when there are duplicates. If a successful READ operation did not occur before the REWRITE operation:

- The file status key, if defined, is set to 94.
- The EXCEPTION/ERROR procedure, if any, is run.
- The REWRITE statement is not processed.

If the last input/output statement was a successfully processed READ statement **with** the NO LOCK phrase:

- The file status key, if defined, is set to 9S.
- The EXCEPTION/ERROR procedure, if any, is run.
- The REWRITE statement is not processed.

The value of the RECORD KEY data item should not have changed since the record was read. By default, if it has changed:

- The file status key, if defined, is set to 21.
- An INVALID KEY condition exists.
- The REWRITE operation does not occur.

If the *NOFS21DUPKY option is in effect, the value of the RECORD KEY data item can be different. Note that this option is provided for compatibility only; its use is not recommended.

Note: The only way to rewrite one of a sequence of records with duplicate keys is to sequentially read each of the duplicate records and rewrite the desired one.

End of IBM Extension

E The value of the RECORD KEY data item must not have been changed since the record was read.

IBM Extension

F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format to operate on.

If an identifier is specified, it must be a character-string of ten characters or less, and it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

If a literal is specified, it must be an uppercase character-string of ten characters or less.

A value of all blanks is treated as though the FORMAT phrase were not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is invoked, if applicable for the file.

End of IBM Extension

G Processed when the value contained in the RECORD KEY of the record to be replaced does not equal the RECORD KEY data item of the last retrieved record from the file.

H Processed when the record specified by the key field in the record area is not found.

J When an INVALID KEY condition exists, the updating operation does not take place. The data in record-name is unaffected.

M Optional when processing a file that has one record format.

O You can specify this combination.

P Allowed when the file is opened for I-O.

S The record to be replaced is specified by the value in the RELATIVE KEY data item. If the file does not contain such a record, an INVALID KEY condition exists.

U1 The INVALID KEY phrase must be specified if no applicable EXCEPTION/ERROR procedure is specified for record-name-1.

An INVALID KEY condition exists when:

- The access mode is sequential, and the value contained in the prime RECORD KEY of the record to be replaced does not equal the value of the prime RECORD KEY data item of the last-retrieved record from the file, or
- The value contained in the prime RECORD KEY does not equal that of any record in the file.

The INVALID KEY phrase must be specified for files in which an applicable USE procedure is not specified.

See "Invalid Key Condition" under "Common Processing Facilities" on page 214 for more information.

U2 After the successful completion of a REWRITE statement with the NOT INVALID KEY phrase, control transfers to the imperative statement associated with the phrase.

IBM Extension

Z The action of this statement can be inhibited at program run time by the inhibit write (INHWRT) parameter of the Override with database file (OVRDBF) CL command. When this parameter is specified, nonzero file status codes are not set for data dependent errors. Duplicate key and data conversion errors are examples of data dependent errors.

See the *CL Reference* for more information on this command.

End of IBM Extension

Sequential Files

For files in the sequential access mode, the last prior input/output statement executed for this file must be a successfully executed READ statement. When the REWRITE statement is executed, the record retrieved by that READ statement is logically replaced.

The number of character positions in record-name-1 must equal the number of character positions in the record being replaced.

The INVALID KEY phrase must not be specified for a file with sequential organization. An EXCEPTION/ERROR procedure may be specified.

Indexed Files

The number of character positions in record-name-1 must equal the number of character positions in the record being replaced.

When the access mode is sequential, the record to be replaced is specified by the value contained in the prime RECORD KEY. When the REWRITE statement is executed, this value must equal the value of the prime record key data item in the last record read from this file.

The INVALID KEY phrase must be specified if an applicable USE AFTER STANDARD EXCEPTION procedure is not specified for the associated file-name.

When the access mode is random or dynamic, the record to be replaced is specified by the value contained in the prime RECORD KEY.

If an invalid key condition exists, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, and the data in record-name-1 is unaffected. (See "INVALID KEY Condition" on page 215.)

Relative Files

The number of character positions in record-name-1 must equal the number of character positions in the record being replaced.

For relative files in sequential access mode, the INVALID KEY phrase must not be specified. An EXCEPTION/ERROR procedure may be specified.

The INVALID KEY phrase must be specified in the REWRITE statement for relative files in the random or dynamic access mode, and for which an appropriate USE AFTER STANDARD EXCEPTION procedure is not specified.

When the access mode is random or dynamic, the record to be replaced is specified in the RELATIVE KEY data item. If the file does not contain the record specified, an invalid key condition exists, and, if specified, the INVALID KEY imperative-statement is executed. (See "INVALID KEY Condition" on page 215.) The updating operation does not take place, and the data in record-name is unaffected.

Transaction Format

The REWRITE statement is used to replace a subfile record that already exists in the subfile.

The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced. A successful read operation on the record must be done prior to the rewrite operation. The record replaced in the subfile is that record accessed by the previous read operation.

FORMAT Phrase

Multiple data records, each with a different format, can be concurrently active for a TRANSACTION file. If the FORMAT phrase is specified, it must specify a valid format name that is defined to the system, and the I-O operation must be performed on a data record of the same format. If the format is

an invalid name or if it does not exist, the FILE STATUS data item, if specified, is set to a value of 9K and the contents of the record area are undefined.

Notes:

1. The record format specified in the FORMAT phrase must be the record format accessed on the previous read operation.
2. Literal-1 or the contents of identifier-2 must be the name of the subfile format accessed on the previous READ.

TERMINAL Phrase

The TERMINAL phrase indicates which program device's subfile is to have a record rewritten. If the TERMINAL phrase is specified, literal-2 or identifier-3 must refer to a work station that has been acquired by the TRANSACTION file. If literal-2 or identifier-3 contains blanks, the TERMINAL phrase has no effect. The program device specified by the TERMINAL phrase must have been acquired, either explicitly or implicitly, and must have a subfile associated with the device.

Literal-2 or identifier-3 must be a valid program device name. Literal-2, if specified, must be nonnumeric and 10 characters or less. Identifier-3, if specified, must refer to an alphanumeric data item, 10 characters or less in length.

If the TERMINAL phrase is omitted from a TRANSACTION file that has acquired multiple program devices, the subfile used is the subfile associated with the last program device from which a READ of the TRANSACTION file was attempted.

The REWRITE statement cannot be used for communications devices. If the REWRITE statement is used for a communications device, the operation fails and a file status of 90 is set.

INVALID KEY Phrase

If, at the time of the rewrite operation, the RELATIVE KEY data item contains a value that does not correspond to the relative record number of the record from the previous read operation, the INVALID KEY condition exists.

The INVALID KEY phrase should be specified for files for which an appropriate USE procedure is not specified. Undesirable results may occur if the INVALID KEY phrase is not specified, and no USE procedure is specified.

NOT INVALID KEY Phrase

After the successful completion of a REWRITE statement with the NOT INVALID KEY phrase, control transfers to the imperative statement associated with the phrase.

END-REWRITE Phrase

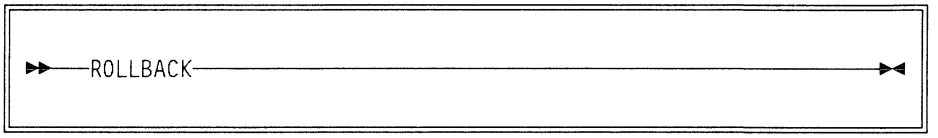
This explicit scope terminator serves to delimit the scope of the REWRITE statement. END-REWRITE permits a conditional REWRITE statement to be nested in another conditional statement. END-REWRITE may also be used with an imperative REWRITE statement. For more information, see "Delimited Scope Statements" on page 207.

IBM Extension

ROLLBACK Statement

The ROLLBACK statement provides a way to cancel one or more changes to database records when the changes should not remain permanent.

Format



When the ROLLBACK statement is executed, any changes made to files under commitment control since the last commitment boundary are removed from the database.[3] A commitment boundary is the previous occurrence of a ROLLBACK or COMMIT statement. If no COMMIT or ROLLBACK has been issued, the commitment boundary is the first OPEN of a file under commitment control. Removal of changes takes place for all files under commitment control in the job, and not just for files under commitment control in the COBOL program that issues the ROLLBACK.

Once the ROLLBACK is successfully executed, all record locks held by the job for files under commitment control are released and the records become available to other jobs.

The ROLLBACK has no effect on files not under commitment control. If a ROLLBACK is executed and there are no files under commitment control, the ROLLBACK is ignored.

A file under commitment control can be opened or closed without affecting the status of changes made since the last commitment boundary. A COMMIT must still be issued to make the changes permanent. A ROLLBACK, when executed, leaves files in the same open or closed state as before execution.

The ROLLBACK statement does *not*:

- modify the I-O-FEEDBACK area for any file
- set a file status value for any file.

For the ROLLBACK statement, the following considerations apply:

- The ROLLBACK statement sets the file position indicator to the pointer's position at the previous commitment boundary. This is important to remember if you are doing sequential processing.

[3] When a file is cleared while being opened for OUTPUT, execution of a ROLLBACK statement does not restore cleared records to the file.

- If no COMMIT statement has been issued since the file was opened, the ROLLBACK statement sets the file position indicator to the pointer's position at the OPEN.
- The file position indicator is undefined after a ROLLBACK if the file is closed with uncommitted changes.

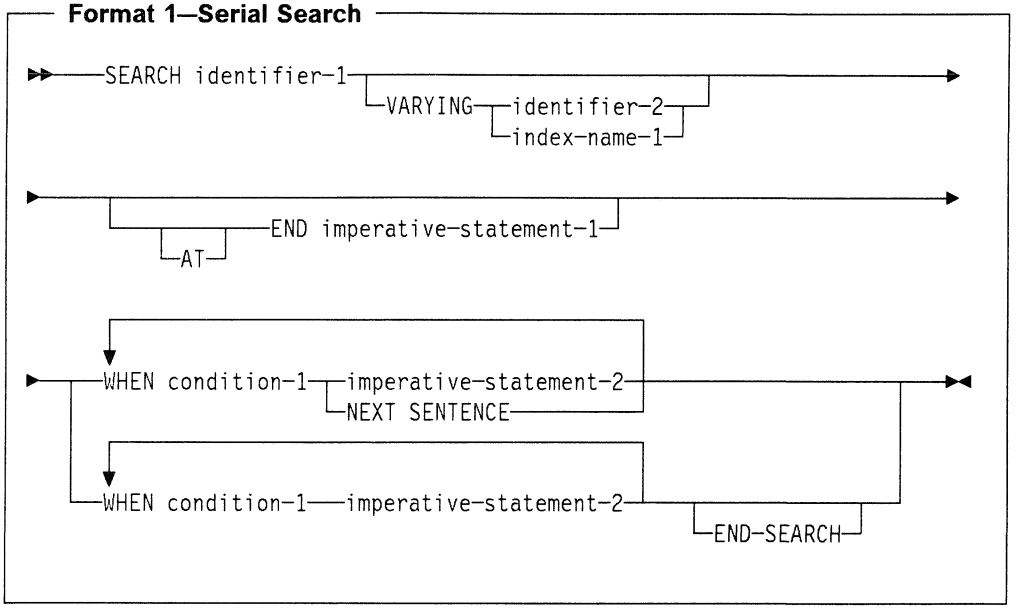
At the end of every job, an implicit ROLLBACK of uncommitted records is automatically done for all files under commitment control. Any uncommitted changes to the database are cancelled.

End of IBM Extension

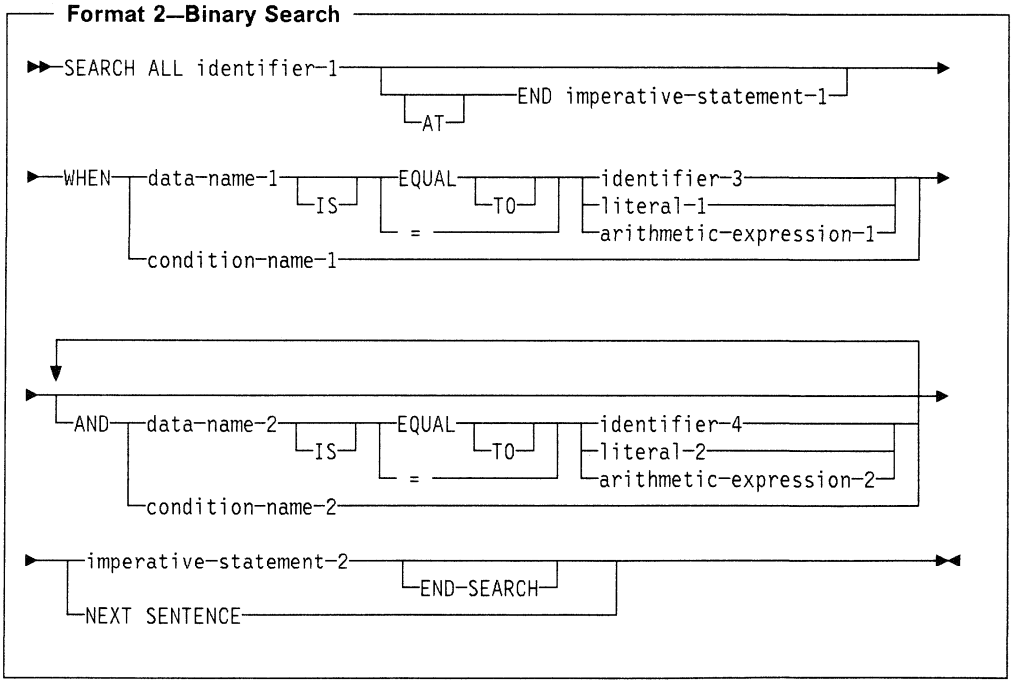
SEARCH Statement

The SEARCH statement searches a table for an element that satisfies the specified condition, and adjusts the associated index to indicate that element.

Format 1—Serial Search



Format 2—Binary Search



identifier-1

Can be a data item subordinate to a data item that contains an OCCURS clause; that is, it can be a part of a multi-dimensional table. In this case, the data description entry must specify an INDEXED BY phrase for each dimension of the table.

Identifier-1 can be an index data item.

Identifier-1 must refer to all occurrences within the table element; that is, it must not be subscripted or reference modified.

The Data Division description of identifier-1 must contain an OCCURS clause with the INDEXED BY phrase. For Format-2, the Data Division description must also contain the KEY IS phrase in its OCCURS clause.

SEARCH statement execution modifies only the value in the index-name associated with identifier-1 (and, if present, of index-name-1 or identifier-2). Therefore, to search an entire two- to seven-dimensional table, it is necessary to execute a SEARCH statement for each dimension. Before each execution, SET statements must be executed to reinitialize the associated index-names.

AT END/WHEN Phrases

After imperative-statement-1 or imperative-statement-2 is executed, control passes to the end of the SEARCH statement, unless imperative-statement-1 or imperative-statement-2 ends with a GO TO statement.

END-SEARCH Phrase

This explicit scope terminator serves to delimit the scope of the SEARCH statement. END-SEARCH permits a conditional SEARCH statement to be nested in another conditional statement. If the END-SEARCH phrase is specified, the NEXT SENTENCE phrase must not be specified.

For more information, see "Delimited Scope Statements" on page 207.

Serial Search

The Format 1 SEARCH statement executes a serial search beginning at the current index setting. When the search begins, if the value of the index-name associated with identifier-1 is not greater than the highest possible occurrence number, the following actions take place:

- The condition(s) in the WHEN phrase are evaluated in the order in which they are written.
- If none of the conditions is satisfied, the index-name for identifier-1 is increased to correspond to the next table element, and step 1 is repeated.
- If upon evaluation, one of the WHEN conditions is satisfied, the search is terminated immediately, and the imperative-statement associated with that condition is executed. The index-name points to the table element that satisfied the condition. If NEXT SENTENCE is specified, control passes to the statement following the closest period.
- If the end of the table is reached (that is, the incremented index-name value is greater than the highest possible occurrence number) without the WHEN condition being satisfied, the search is terminated, as described in the next paragraph.

If, when the search begins, the value of the index-name associated with identifier-1 is greater than the highest possible occurrence number, the search immediately ends, and, if specified, the AT END imperative-statement is executed. If the AT END phrase is omitted, control passes to the next statement after the SEARCH statement.

VARYING Phrase

index-name-1

One of the following actions applies:

- If index-name-1 is an index for identifier-1, this index is used for the search. Otherwise, the first (or only) index-name is used.
- If index-name-1 is an index for another table element, then the first (or only) index-name for identifier-1 is used for the search; the occurrence number represented by index-name-1 is increased by the same amount as the search index-name and at the same time.

When the VARYING index-name-1 phrase is omitted, the first (or only) index-name for identifier-1 is used for the search.

identifier-2

Must be either an index data item or an elementary integer item. During the search, one of the following actions applies:

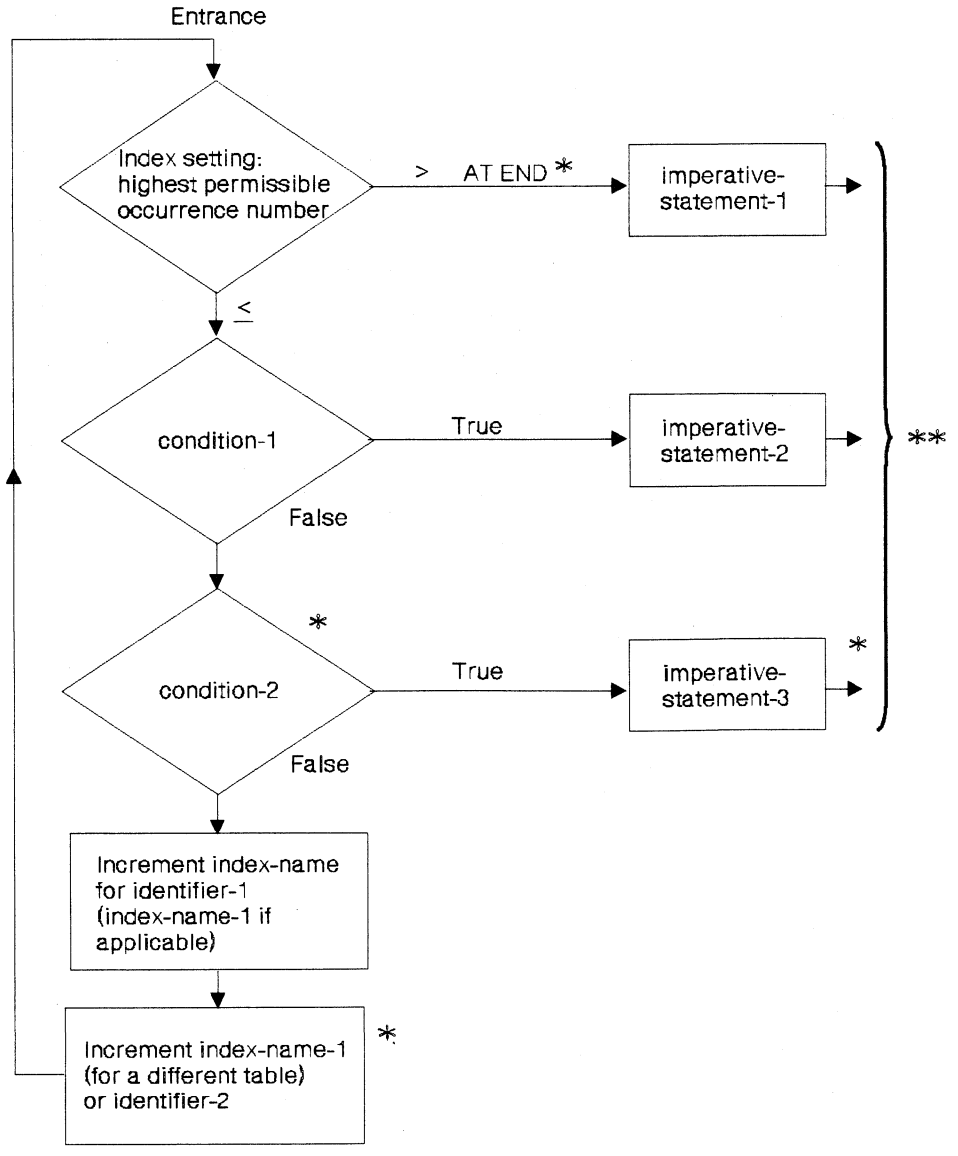
- If identifier-2 is an index data item, then, whenever the search index is increased, the specified index data item is simultaneously increased by the same amount.
- If identifier-2 is an integer data item, then, whenever the search index is increased, the specified data item is simultaneously increased by 1.

WHEN Phrase

condition-1, condition-2

May be any condition described under "Conditional Expressions" on page 189.

Figure 25 illustrates a Format 1 SEARCH operation containing two WHEN phrases.



* These operations are included only when called for in the statement.

** Control transfers to the next sentence, unless the imperative statement ends with a GO TO statement.

Figure 25. Format 1 SEARCH with Two WHEN Phrases

Binary Search

The Format 2 SEARCH ALL statement executes a binary search. The search index need not be initialized by SET statements, because its setting is varied during the search operation so that its value is at no time less than the value of the first table element, nor ever greater than the value of the last table element. The index used is always that associated with the first index-name specified in the OCCURS clause.

identifier-1

Can be a data item subordinate to a data item that contains an OCCURS clause; that is, it can be a part of a two- to seven-dimensional table. In this case, the data description entry must specify an INDEXED BY phrase for each dimension of the table.

Before the search takes place, the values of all indexes should be set for higher dimensions of the table to define a specific table of identifier-1 elements.

Identifier-1 must refer to all occurrences within the table element; that is, it must not be subscripted or indexed.

Identifier-1 cannot be a pointer data item.

The Data Division description of identifier-1 must contain an OCCURS clause with the INDEXED BY option.

AT END

The AT END imperative-statement is executed.

If the AT END phrase is not specified, control passes to the next statement after the SEARCH statement.

In either case, the final setting of the index is not predictable.

WHEN Phrase

If the WHEN phrase **cannot** be satisfied for any setting of the index within this range, the search is unsuccessful.

If the WHEN option **can** be satisfied, control passes to imperative-statement-2, and the index contains the value indicating the occurrence that allowed the WHEN condition(s) to be satisfied.

condition-name

Each condition-name specified must have only a single value, and each must be associated with an ASCENDING/DESCENDING KEY identifier for this table element.

data-name

Must specify an ASCENDING/DESCENDING KEY data item in the identifier-1 table element and must be indexed by the first identifier-1 index-name, along with other indexes or literals, as required. Each data-name may be qualified.

identifier-3, identifier-4

Must not be an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is indexed by the first index-name for identifier-1.

Must not be a pointer data item.

arithmetic-expression

May be any of the expressions defined under "Arithmetic Expressions" on page 187, with the following restriction: Any identifier in the arithmetic-expression must not be an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is indexed by the first index-name for identifier-1.

When an ASCENDING/DESCENDING KEY data item is specified, explicitly or implicitly, in the WHEN phrase, all preceding ASCENDING/DESCENDING KEY data-names for identifier-1 must also be specified.

The results of a SEARCH ALL operation are predictable **only** when:

- The data in the table is ordered according to the ASCENDING/DESCENDING KEY phrase
- The contents of the ASCENDING/DESCENDING keys specified in the WHEN clause provide a unique table reference.

Search Statement Considerations

Index data items cannot be used as subscripts, because of the restrictions on direct reference to them.

The use of a direct indexing reference together with a relative indexing reference for the same index-name allows reference to two different occurrences of a table element for comparison purposes.

When the object of the VARYING option is an index-name for another table element, one Format 1 SEARCH statement steps through two table elements at once.

To ensure correct execution of a SEARCH statement for a variable-length table, make sure the object of the OCCURS DEPENDING ON clause (data-name-1) contains a value that specifies the current length of the table.

The scope of a SEARCH statement may be terminated by any of the following:

- An END-SEARCH phrase at the same level of nesting
- A separator period
- An ELSE or END-IF phrase associated with a previous IF statement.

SEARCH Example

The following example searches an inventory table for items that match those from input data. The key is ITEM-NUMBER.

```
.. 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7

 DATA DIVISION.
 FILE SECTION.
 FD  SALES-DATA
     BLOCK CONTAINS 1 RECORDS
     RECORD CONTAINS 80 CHARACTERS
     LABEL RECORDS STANDARD
     DATA RECORD IS SALES-REPORTS.
 01  SALES-REPORTS          PIC X(80).
 FD  PRINTED-REPORT
     BLOCK CONTAINS 1 RECORDS
     RECORD CONTAINS 132 CHARACTERS
     LABEL RECORDS OMITTED
     DATA RECORD IS PRINTER-OUTPUT.
 01  PRINTER-OUTPUT         PIC X(132).
 FD  INVENTORY-DATA
     BLOCK CONTAINS 1 RECORDS
     RECORD CONTAINS 40 CHARACTERS
     LABEL RECORDS STANDARD
     DATA RECORD IS INVENTORY-RECORD.
 01  INVENTORY-RECORD.
     03  I-NUMBER           PIC 9(4).
     03  INV-ID             PIC X(26).
     03  I-COST             PIC 9(8)V99.
 WORKING-STORAGE SECTION.
 01  EOF-SW                 PIC X      VALUE "N".
 01  EOF-SW2                PIC X      VALUE "N".
 01  SUB1                   PIC 99.
 01  RECORDS-NOT-FOUND      PIC 9(5)    VALUE ZEROS.
 01  TOTAL-COSTS            PIC 9(10)   VALUE ZEROS.
 01  HOLD-INPUT-DATA.
     03  INVENTORY-NUMBER  PIC 9999.
     03  PURCHASE-COST     PIC 9(4)V99.
     03  PURCHASE-DATE     PIC 9(6).
     03  FILLER            PIC X(64).
 01  PRINTER-SPECS.
     03  PRINT-LINE.
         05  OUTPUT-ITEM-NUMBER       PIC ZZZ9.
         05  FILLER                   PIC X(48) VALUE SPACES.
         05  TOTAL-COSTS-0            PIC $(8).99.
 01  PRODUCT-TABLE.
     05  INVENTORY-NUMBERS  OCCURS 50 TIMES
                            ASCENDING KEY ITEM-NUMBER
                            INDEXED BY INDEX-1.
             07  ITEM-NUMBER           PIC 9(4).
             07  ITEM-DESCRIPTION      PIC X(26).
             07  ITEM-COST             PIC 9(8)V99.
```

```
.. 1  ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ...7
 PROCEDURE DIVISION.
 100-START-IT.
     OPEN INPUT SALES-DATA INVENTORY-DATA OUTPUT PRINTED-REPORT.
     MOVE HIGH-VALUES TO PRODUCT-TABLE.
     PERFORM READ-INVENTORY-DATA.
 LOAD-TABLE-ROUTINE.
     PERFORM LOAD-IT VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 > 50
                     OR EOF-SW2 = "Y".
     PERFORM 110-READ-IT.
 200-MAIN-ROUTINE.
     PERFORM PROCESS-DATA UNTIL EOF-SW = "Y".
     MOVE TOTAL-COSTS TO TOTAL-COSTS-0.
     PERFORM WRITE-REPORT THRU WRITE-REPORT-EXIT.
     DISPLAY "RECORDS NOT FOUND - " RECORDS-NOT-FOUND
       UPON MYTUBE.
     STOP RUN.
 PROCESS-DATA.
     SEARCH ALL INVENTORY-NUMBERS
       AT END PERFORM KEY-NOT-FOUND THRU NOT-FOUND-EXIT
       WHEN ITEM-NUMBER (INDEX-1) = INVENTORY-NUMBER
          MOVE ITEM-NUMBER (INDEX-1) TO OUTPUT-ITEM-NUMBER
          MOVE ITEM-COST (INDEX-1) TO TOTAL-COSTS-0
          ADD ITEM-COST (INDEX-1) TO TOTAL-COSTS
          PERFORM WRITE-REPORT THRU WRITE-REPORT-EXIT.
     PERFORM 110-READ-IT.
 KEY-NOT-FOUND.
     ADD 1 TO RECORDS-NOT-FOUND.
 NOT-FOUND-EXIT.
     EXIT.
 LOAD-IT.
     MOVE INVENTORY-RECORD TO INVENTORY-NUMBERS (SUB1).
     PERFORM READ-INVENTORY-DATA.
 WRITE-REPORT.
     WRITE PRINTER-OUTPUT FROM PRINTER-SPECS.
 WRITE-REPORT-EXIT.
     EXIT.
 READ-INVENTORY-DATA.
     READ INVENTORY-DATA
       AT END MOVE "Y" TO EOF-SW2.
 110-READ-IT.
     READ SALES-DATA INTO HOLD-INPUT-DATA
       AT END MOVE "Y" TO EOF-SW.
```

SET Statement

The SET statement establishes reference points for table handling operations by doing one of the following:

1. Placing values associated with table elements into indexes associated with index-names
2. Incrementing or decrementing an occurrence number
3. Setting the status of an external switch to ON or OFF, or
4. Moving data to condition names to make conditions true.
5. Setting addresses of pointer data items.

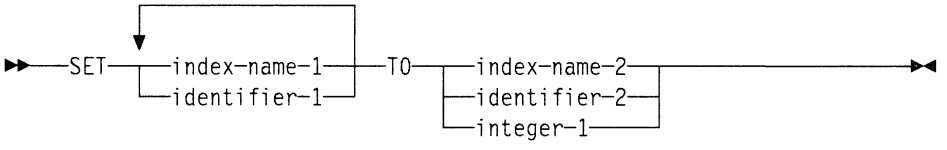
Index-names are related to a given table through the INDEXED BY phrase of the OCCURS clause; they are not further defined in the program.

When the sending and receiving fields in a SET statement share part of their storage (that is, the operands overlap), the result of the execution of such a SET statement is undefined.

Format 1: TO Phrase

When this form of the SET statement is executed, the current value of the receiving field is replaced by the value of the sending field (with conversion).

Format 1—TO Phrase



index-name-1, identifier-1
Receiving fields.

Must name either index data items or elementary numeric integer items.

index-name-2
Sending field.

The value before the SET statement is executed must correspond to the occurrence number of its associated table.

identifier-2
Sending field.

Must name either an index data item or an elementary numeric integer item.

integer-1
Sending field.

Must be a positive integer.

Execution of the Format 1 SET statement depends upon the type of receiving field, as follows:

- Index-name receiving fields (index-name-1, index-name-2, and so on) with one exception are converted to a displacement value representing the occurrence number indicated by the sending field. To be valid, the resulting index-name value must correspond to an occurrence number in its associated table element. For the one exception, when the sending field is an index data item, the value in the index data item is placed in the index-name without change.
- Index data item receiving fields (identifier-1, identifier-2, and so on) are set equal to the contents of the sending field (which must be either an index-name or an index data item); no conversion takes place. A numeric integer or literal sending field must not be specified.
- Integer data item receiving fields (identifier-1, identifier-2, and so on) are set to the occurrence number associated with the sending field, which must be an index-name. An integer data item, an index data item, or a literal sending field must not be specified.

Table 48 shows valid combinations of sending and receiving fields in a Format 1 SET statement.

Table 48. Sending and Receiving Fields for Format 1 SET Statement

Sending Field	**Receiving Field**		
	Index-name	**Index Data Item**	**Integer Data Item**
Index-name	Valid	Valid*	Valid
Index Data Item	Valid*	Valid*	—
Integer Data Item	Valid	—	—
Integer Literal	Valid	—	—
*No conversion takes place			

Receiving fields are acted upon in the left-to-right order in which they are specified. Any subscripting or indexing associated with an identifier's receiving field is evaluated immediately before the field is acted upon.

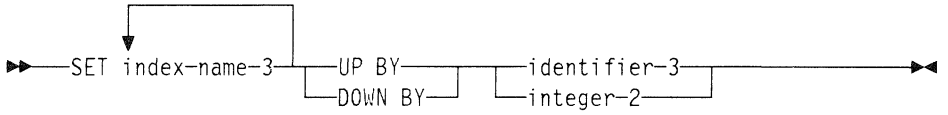
The value used for the sending field is the value at the beginning of SET statement execution.

The value for an index-name after execution of a SEARCH or PERFORM statement may be undefined; therefore, a Format 1 SET statement should reinitialize such index-names before other table-handling operations are attempted.

Format 2: UP BY/DOWN BY Phrase

When this form of the SET statement is executed, the value of the receiving field is increased (UP BY) or decreased (DOWN BY) by a value that corresponds to the value in the sending field.

Format 2—UP BY/DOWN BY Phrase



The **receiving field** may be specified by index-name-3. This index-name value both before and after the SET statement execution must correspond to the occurrence numbers in an associated table.

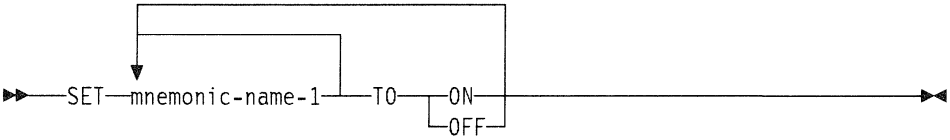
The **sending field** may be specified as identifier-3, which must be an elementary integer data item, or as integer-2, which must be an integer.

When the Format 2 SET statement is executed, the contents of the receiving field are increased (UP BY) or decreased (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of identifier-3 or integer-2. Receiving fields are acted upon in the left-to-right order in which they are specified. The value of the incrementing or decrementing field at the beginning of SET statement execution is used for all receiving fields.

Format 3: ON/OFF Phrase

When this form of the SET statement is executed, the status of each external switch associated with the specified mnemonic-name is turned ON or OFF.

Format 3—ON/OFF Phrase



mnemonic-name

Must be associated with an external switch, the status of which can be altered.

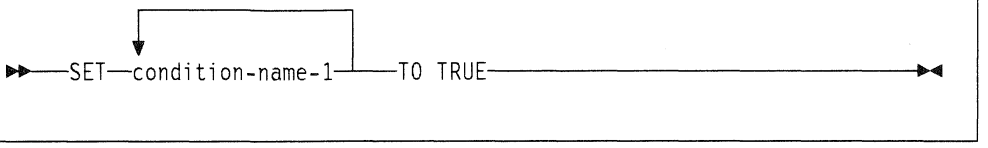
For Format 4 each mnemonic-name must be associated with an external switch, the status of which can be altered. The only external switches allowed are the UPSI switches, UPSI-0 through UPSI-7.

The status of each external switch 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.

Format 4: TO TRUE Phrase

When this form of the SET statement is executed, the value associated with a condition-name is placed in its conditional variable according to the rules of the VALUE clause.

Format 4—TO TRUE Phrase



condition-name-1
Must be associated with a conditional variable.

If more than one literal is specified in the VALUE clause of condition-name-1, its associated conditional variable is set equal to the first literal.

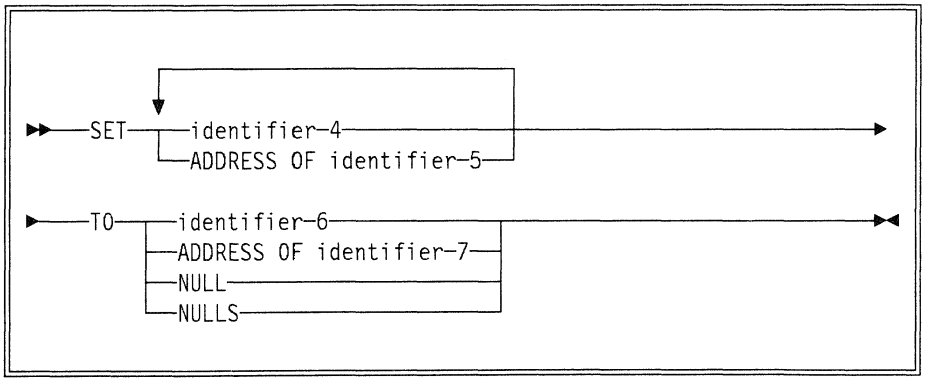
If multiple condition-names are specified, the results are the same as if a separate SET statement had been written for each condition-name in the same order in which they are specified in the SET statement.

IBM Extension

Format 5: Pointer Data Item

When this form of the SET statement is executed, the current value of the receiving field is replaced by the address value contained in the sending field.

Format 5—Pointer Data Item Phrase



identifier-4
Receiving fields.

Must be described as USAGE IS POINTER.

ADDRESS OF identifier-5
Receiving fields.

This is the ADDRESS OF special register.

Must be a level-01 or level-77 item defined in the Linkage section. It is set to the value of the operand specified in the TO phrase. It cannot be subscripted or reference modified.

identifier-6
Sending field.

Must be described as USAGE IS POINTER.

Must not contain an address within the program's own Working-storage or File section.

ADDRESS OF identifier-7
Sending field.

Must name an item in the Linkage section of any level except 66 or 88.

ADDRESS OF identifier-7 contains the address of the identifier, rather than its contents. Identifier-7 can be subscripted, reference modified, or both.

NULL
NULLS
Sending field.

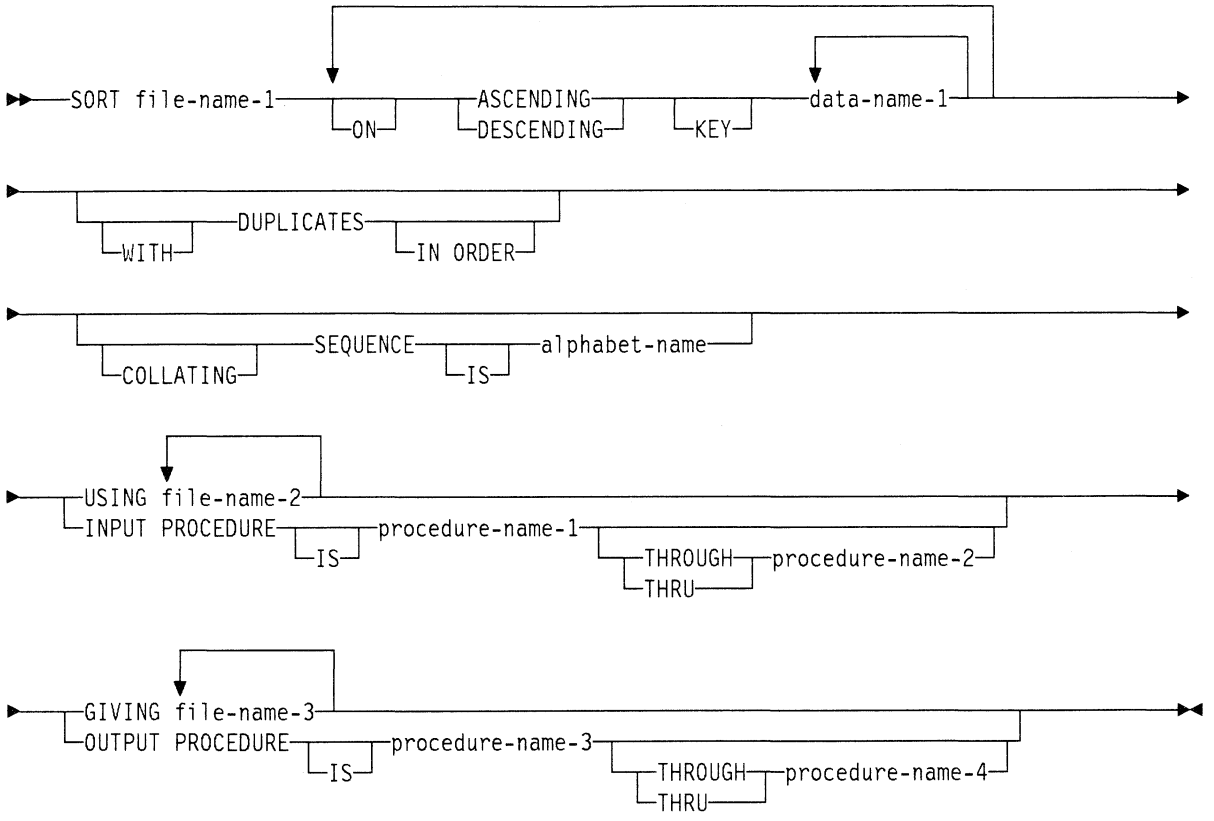
Sets the receiving field to contain the value of an invalid address.

End of IBM Extension

SORT Statement

The SORT statement accepts records from one or more files, sorts them according to the specified key(s), and makes the sorted records available either through an OUTPUT PROCEDURE or in an output file. See also "MERGE Statement" on page 316.

Format



file-name-1
The name given in the SD entry that describes the records being sorted.

Null-capable fields are supported, but null values are not. Null values result in a file status of 90.

ASCENDING/DESCENDING KEY Phrase

This phrase specifies that records are to be processed in ascending or descending sequence (depending on the phrase specified), based on the specified sort keys.

data-name-1

Specifies a KEY data item on which the sort will be based. Each such data-name must identify a data item in a record associated with **file-name-1**. The following rules apply:

1. A specific KEY data item must be physically located in the same position and have the same data format in each input file. However, it need not have the same data-name.
2. If file-name-1 has more than one record description, then the KEY data items need be described in only one of the record descriptions.
3. KEY data items must be fixed-length items.
4. KEY data items must not contain an OCCURS clause or be subordinate to an item that contains an OCCURS clause.
5. The total length of the KEY data item must not exceed 256 bytes.
6. KEY data items can be qualified or reference modified; they cannot be subscripted or indexed.

SORT lists the KEY data items from left to right in order of decreasing significance, no matter 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, and so forth.

The direction of the sorting operation depends on the specification of the ASCENDING or DESCENDING key words as follows:

- When ASCENDING is specified, the sequence is from the lowest key value to the highest key value.
- When DESCENDING is specified, the sequence is from the highest key value to the lowest.
- If the KEY data item is alphabetic, alphanumeric, alphanumeric-edited, or numeric-edited, the sequence of key values depends on the collating sequence used (see "COLLATING SEQUENCE Phrase" on page 401).
- The key comparisons are performed according to the rules for comparison of operands in a relation condition (see "Relation Condition" under "Conditional Expressions" on page 189).

DUPLICATES Phrase

If the DUPLICATES phrase is specified, and the contents of all the key elements associated with one record are equal to the corresponding key elements in one or more other records, the order of these records is as follows:

- The order of the associated input files as specified in the SORT statement. Within a given file the order is that in which the records are accessed from that file.
- The order in which these records are released by an input procedure, when an input procedure is specified.

If the DUPLICATES phrase is not specified, the order of these records is undefined.

COLLATING SEQUENCE Phrase

This phrase specifies the collating sequence to be used in nonnumeric comparisons for the KEY data items in this sorting operation.

alphabet-name
Must be specified in the alphabet-name clause of the SPECIAL-NAMES paragraph. Any one of the alphabet-name clause options may be specified. See "SPECIAL-NAMES Paragraph" on page 57 for a list of alphabet-name clause options and their meanings.

When the COLLATING SEQUENCE phrase is omitted, the PROGRAM COLLATING SEQUENCE clause (if specified) in the OBJECT-COMPUTER paragraph specifies the collating sequence to be used.

When both the COLLATING SEQUENCE phrase and the PROGRAM COLLATING SEQUENCE clause are omitted, the EBCDIC collating sequence is used.

USING Phrase

file-name-2, . . .
The input files.

When the USING phrase is specified, all the records in **file-name-2**, . . . (that is, the input files) are transferred automatically to file-name-1. At the time the SORT statement is executed, these files must not be open; the compiler opens, reads, makes records available, and closes these files automatically. If EXCEPTION/ERROR procedures are specified for these files, the compiler makes the necessary linkage to these procedures. The input files must be sequential, relative or indexed files.

All input files must specify sequential or dynamic access mode, and must be described in FD entries in the Data Division.

INPUT PROCEDURE Phrase

This phrase specifies the name of a procedure that is to select or modify input records before the sorting operation begins.

procedure-name-1
Specifies the first (or only) section or paragraph in the input procedure.

procedure-name-2
Specifies the last section or paragraph in the input procedure.

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.

If an input procedure is specified, control is passed to the input procedure before file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last statement in the input procedure. When control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted.

GIVING Phrase

file-name-3, . . .
The output files.

When the GIVING phrase is specified, all the sorted records in file-name-1 are automatically transferred to the output files (file-name-3, . . .). At the time the SORT statement is executed, this file must not be open.

For each of the files referenced by file-name-3, the execution of the SORT statement causes the following actions to be taken:

- The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase has been executed.
- The sorted 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. The records overwrite the previous contents, if any, of the file.

IBM Extension

If file-name-1 is a logical database file, the records are added to the end of the file.

End of IBM Extension

If the file referenced by file-name-3 is an INDEXED file then the associated key data-name for that file must have an ASCENDING KEY phrase in the SORT statement. This same data-name must occupy the identical character positions in its record as the data item associated with the prime record key for the file.

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', and so on. After execution of the SORT statement, the content of the relative key data item indicates the last record returned to the file.

- The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

Note: When duplicate keys are found when writing to an indexed file, the SORT will terminate and the sorted data in all GIVING files will be incomplete.

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.

All output files must specify sequential or dynamic access mode, and must be described in FD entries in the Data Division.

The output file must be an indexed, relative or sequential file.

The output file should also be created without a keyed sequence access path. When the output file has such a path, the MERGE statement cannot override the collating sequence defined in the data description specifications (DDS).

OUTPUT PROCEDURE Phrase

This phrase specifies the name of a procedure that is to select or modify output records from the sorting operation.

procedure-name-3
Specifies the first (or only) section or paragraph in the output procedure.

procedure-name-4
Identifies the last section or paragraph in the output procedure.

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 file-name-1. The range of the output procedure includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements within the output procedure. The range also includes 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 include any MERGE, RELEASE, or SORT statement.

If an output procedure is specified, control passes to it after file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism after the last statement in the output procedure, and when control passes that statement, the return mechanism terminates the sort and 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.

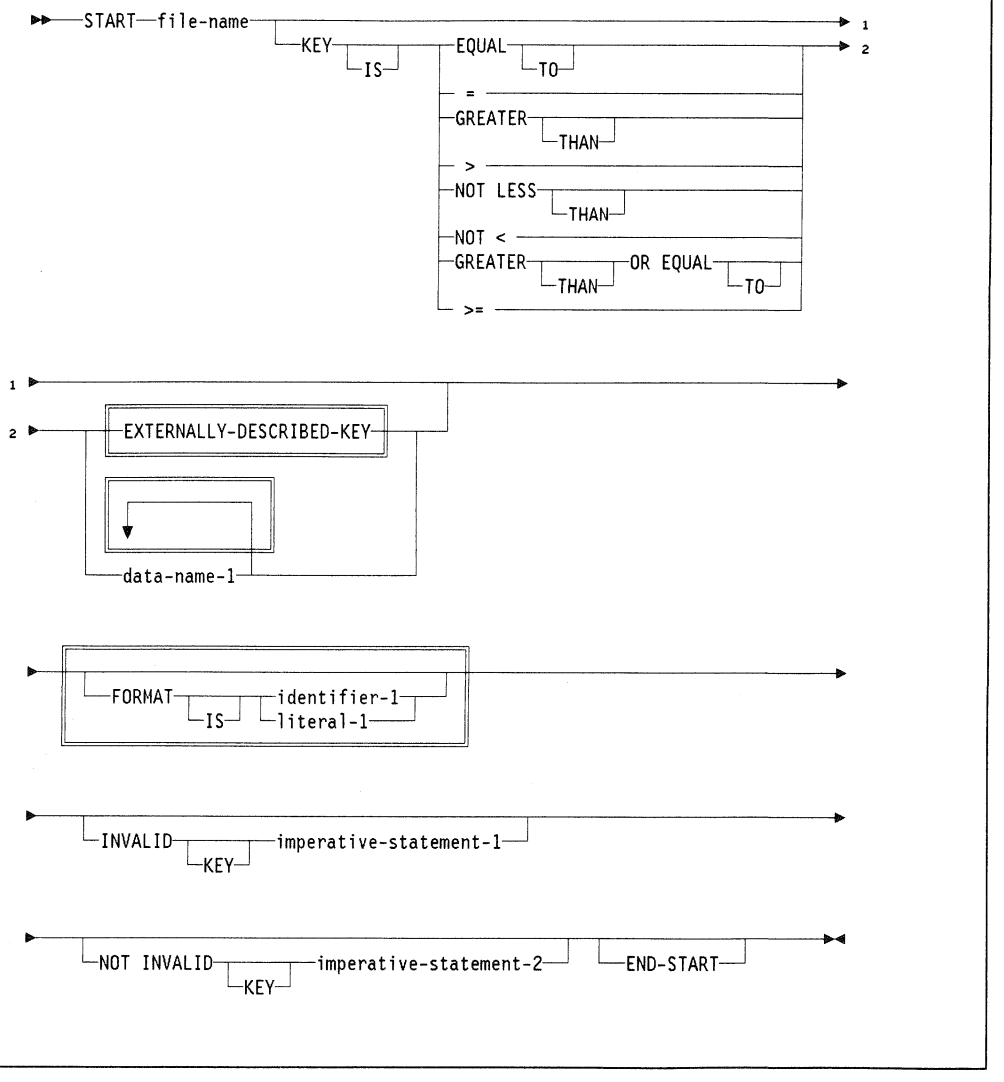
Note: The INPUT and OUTPUT PROCEDURE phrases are similar to those for a basic PERFORM statement. For example, if you name a procedure in an OUTPUT PROCEDURE phrase, that procedure is executed during the sorting operation just as if it were named in a PERFORM statement. As with the PERFORM statement, execution of the procedure ends after the last statement executes. The last statement in an input or output procedure can be the EXIT statement (see "EXIT Statement" on page 297).

START Statement

The START statement provides a means of positioning within an indexed or relative file for subsequent sequential record retrieval. This positioning is achieved by comparing the key values of records in the file with the value you place in the RECORD KEY portion of a file's record area (for an indexed file), or in the RELATIVE KEY data item (for a relative file) prior to execution of the START statement.

Note: When the START statement is executed, the associated indexed or relative file must be open in INPUT or I-O mode.

Format



file-name-1

Must name a file with sequential or dynamic access. File-name-1 must be defined in an FD entry in the Data Division, and must not name a sort file.

KEY Phrase

When the KEY phrase is specified, the file position indicator is positioned at the logical record in the file whose key field satisfies the comparison.

When the KEY phrase is not specified, KEY IS EQUAL (to the prime record key) is implied.

data-name-1
Can be qualified or reference modified, but it cannot be subscripted.

IBM Extension

Multiple data-names may be specified. All data-names, following the initial data-name, are syntax-checked only.

End of IBM Extension

When the START statement is executed, a comparison is made between the current value in the key data-name and the corresponding key field in the file's index.

If the FILE STATUS clause is specified in the FILE-CONTROL entry, the associated status key is updated when the START statement is executed. (See "Status Key" on page 214.)

INVALID KEY Phrase

If the comparison is not satisfied by any record in the file, an invalid key condition exists; the value of the file position indicator is undefined, and (if specified) the INVALID KEY imperative statement runs. (See "INVALID KEY Condition" on page 215.)

The INVALID KEY phrase must be specified if no EXCEPTION/ERROR procedure is explicitly or implicitly specified for this file.

END-START Phrase

This explicit scope terminator serves to delimit the scope of the START statement. END-START permits a conditional START statement to be nested in another conditional statement. END-START may also be used with an imperative START statement.

For more information, see "Delimited Scope Statements" on page 207.

Indexed Files

When the KEY phrase is specified, the key data item used for the comparison is data-name.

When the KEY phrase is not specified, the key data item used for the EQUAL TO comparison is the prime RECORD KEY. When START statement execution is successfully completed, the RECORD KEY becomes the key of reference for subsequent READ statements.

data-name-1
Can be any of the following:

- The prime RECORD KEY.

- An alphanumeric data item within a record description for a file whose leftmost character position corresponds to the leftmost character position of that record key; it may be qualified. The data item must be less than or equal to the length of the record key for the file.

The file position indicator points to the first record in the file whose key field satisfies the comparison. If the operands in the comparison are of unequal lengths, the comparison proceeds as if the longer field were truncated on the right to the length of the shorter field. All other numeric and nonnumeric comparison rules apply, except that the PROGRAM COLLATING SEQUENCE clause, if specified, has no effect.

When START statement execution is successful, the RECORD KEY with which data-name-1 is associated becomes the key of reference for subsequent READ statements.

When START statement execution is unsuccessful, the key of reference is undefined.

Relative Files

When the KEY phrase is specified, data-name-1 must specify the RELATIVE KEY.

Whether or not the KEY phrase is specified, the key data item used in the comparison is the RELATIVE KEY data item. When START statement execution is successful, the file position indicator points to the logical record in the file whose key satisfies the comparison, and this key becomes the reference for subsequent READ statements.

When START statement execution is unsuccessful, the key of reference and the file position indicator are undefined.

START Statement Considerations

The following tables illustrate organization, access, and device considerations for the START statement. The letter codes used in the tables are defined in the section following the tables.

Table 49. Sequential Organization

Device	**Any**
Access	**SEQUENTIAL**
START Verb	Not Allowed

Table 50. Relative Organization

Device	**DISK**			**DATABASE**		
Access	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**
START Verb	I,P,A	—	I,P,A	I,P,A	—	I,P,A
KEY IS	O,E	—	O,E	O,E	—	O,E
INVALID KEY	O,U1,D	—	O,U1,D	O,U1,D	—	O,U1,D
NOT INVALID KEY	O,U2	—	O,U2	O,U2	—	O,U2
FORMAT	—	—	—	—	—	—

Table 51. Indexed Organization						
Device	**DISK**			**DATABASE**		
Access	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**
START Verb	I,P,B	—	I,P,B	I,P,B,K	—	I,P,B,K
KEY IS	O,G	—	O,G	O,G	—	O,G
INVALID KEY	O,U1,D	—	O,U1,D	O,U1,D	—	O,U1,D
NOT INVALID KEY	O,U2	—	O,U2	O,U2	—	O,U2
FORMAT	—	—	—	O,F,J	—	O,F,J

Letter Code **Meaning**

— Combination is not valid.

A When the KEY phrase is not specified, the file position indicator is set to the record in the file with a key (relative record number) equal to the RELATIVE KEY data item.

B When the KEY phrase is not specified, the file position indicator is set to the record with a key equal to the value contained in the RECORD KEY data item.

D If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists. The value of the file position indicator is undefined, and the INVALID KEY imperative statement, if specified, is processed.

The INVALID KEY phrase must be specified if no EXCEPTION/ERROR procedure is explicitly or implicitly specified for this file.

See "INVALID KEY Condition" under "Common Processing Facilities" on page 214 for more information.

E When the KEY phrase is specified, data-name-1 must specify the RELATIVE KEY. The file position indicator is positioned to the first logical record currently existing in the file with a key (relative record number) that satisfies the comparison with the RELATIVE KEY data item.

When the KEY phrase is not specified, KEY IS EQUAL (to the prime record key) is implied.

Data-name-1 may be qualified; it may not be subscripted.

When the START statement is executed, a comparison is made between the current value in the key data-name and the corresponding key field in the file's records.

If the FILE STATUS clause is specified in the FILE-CONTROL entry, the associated status key is updated when the START statement is executed. (See "Status Key" under "Common Processing Facilities" on page 214.)

IBM Extension

F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format to operate on.

If an identifier is specified, it must be a character-string of ten characters or less, and it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

If a literal is specified, it must be an uppercase character-string of ten characters or less.

A value of all blanks is treated as though the FORMAT phrase were not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is invoked, if applicable for the file.

End of IBM Extension

G When the KEY phrase is specified, the search argument used for the comparison is data-name-1, which can be:

- The RECORD KEY itself.
- An alphanumeric data item within a record description for the file with a leftmost character position that corresponds to the leftmost character position of the key field in the record area. This data item must be less than or equal to the length of the record key for the file.

  This data item can be qualified or reference modified. If the key itself is not used, the leftmost character position plus the reference modification starting position must correspond to the leftmost character position of the key field.

Note: If the RECORD KEY is defined as COMP, COMP-3, or COMP-4, the key data item must be the RECORD KEY itself. A partial key field in the record area cannot be used.

The file position indicator is positioned to the first record in the file with a record key for a format that satisfies the comparison. If the operands in the comparison are of unequal length, the comparison proceeds as if the longer field were truncated on the right to the length of the shorter field. All other numeric and nonnumeric comparison rules apply, except that the PROGRAM COLLATING SEQUENCE, if specified, has no effect.

IBM Extension

For a file that specified RECORD KEY IS EXTERNALLY-DESCRIBED-KEY, the following additional considerations apply:

- The reserved word EXTERNALLY-DESCRIBED-KEY can be specified. This indicates that the complete key field in the record area should be used in the comparison.
- A series of data names can be specified. This allows a partial key field in the record area to be used (generic START). These data names must follow the following rules:
  - All except the last of the data names specified must be a record key for a format that was copied in for the file. The record format in which they are contained does not have to be the one that can be specified by the FORMAT phrase.

- The order of these data names (key fields) must match the order of the keys as defined in DDS; that is, they must be specified from most significant field to least significant.
- The total number of data names cannot exceed the number of key fields defined for that record format.
- If the last data name specified in the series is not a key field in the record area, it must have its left byte occupy the same space as the key field that is defined at that relative position. If the key field in the record area at this position is a COMP, COMP-3, or COMP-4 field, only the key field itself can be used as the data name.
- Only the last key can be reference modified, and the reference modification starting position must equal 1.

• The following table shows the action between the KEY IS phrase and the FORMAT phrase:

FORMAT Phrase	KEY Phrase		
	Data-Name Series	Omitted	EXTERNALLY-DESCRIBED-KEY
Yes	G1, G2	G3, G4	G3, G2
No	G1, G5	G6, G7	G6, G5

G1 The search argument is built using the specified data items.

G2 The file position indicator is set to the first record in the file of the format specified with a record key that satisfies the comparison specified in the key phrase.

G3 The search argument is built using the key fields in the record area for the format specified in the FORMAT phrase.

G4 The file position indicator is set to the first record in the file of the specified format with a record key equal to the search argument.

G5 The file position indicator is set to the first record in the file with a common key for the file that satisfies the comparison specified in the KEY phrase. If there is no common key, the file position indicator is set to the first record in the file.

G6 The search argument is built using the key fields in the record area for the first record format for the file as defined in the program.

G7 The file position indicator is set to the first record in the file with a common key for the file that is equal to the search argument. If there is no common key, the file position indicator is set to the first record in the file.

End of IBM Extension

I Allowed when the file is opened for INPUT.

J If specified, the file position indicator is set to the first record of the specified record format that satisfies the comparison. If omitted, the current record pointer is set to the first record of any format that satisfies the comparison.

See the table in G above for a description of how this interacts with EXTERNALLY-DESCRIBED-KEY and the KEY IS phrase.

IBM Extension

K The meaning of the comparison can be affected by the type of key fields in the record area defined for the file. Key fields on this system can be defined as multiple fields, each of which can be in ascending or descending sequence. The system establishes a sequence (keyed sequence access path) for the records based on the values contained in the record key for the format and the sequencing specified in DDS. When a START statement is processed, the request is interpreted as follows:

COBOL Comparison	System Result
GREATER THAN	AFTER
NOT LESS THAN	EQUAL TO or AFTER

For example, when a statement is processed using the comparison of GREATER THAN, a search is made of these sequenced records for the first record after the search argument specified by the START statement. If the file was sequenced using descending keys, the file position indicator would point to a record with a key less than the one specified and not greater than that specified in the START statement.

End of IBM Extension

O You can specify this combination.

P Allowed when the file is opened for I-O.

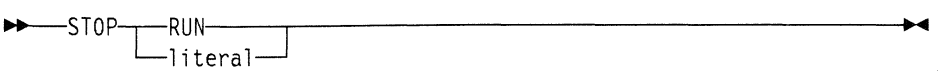
U1 The INVALID KEY phrase must be specified for files in which an appropriate USE procedure is not specified.

U2 After successful completion of a START statement with the NOT INVALID KEY phrase, control transfers to the imperative statement associated with the phrase.

STOP Statement

The STOP statement halts execution of the object program either permanently or temporarily.

Format



literal
May be numeric or nonnumeric, and may be any figurative constant except ALL literal. If the literal is numeric, it must be an unsigned integer.

Note: STOP literal is an obsolete element and is to be deleted from the next revision of the ANSI Standard.

When STOP literal is specified, the literal is communicated to the system operator for batch jobs and to the work station for interactive jobs. Program execution is suspended. Execution is resumed only after operator intervention.

The operator response determines whether the run unit continues at the next executable statement in the sequence, or a STOP RUN is executed.

Operator Response	Action
G (default)	Continue at next instruction.
C	Terminate execution of the run unit. Escape message LBE9001 is issued to the caller of the COBOL run unit. For batch jobs, the job is canceled if the CNLSEV parameter for the job contains a value that is less than or equal to the severity of the message.

The output of the STOP literal contains the program-name followed by the literal.

If the literal cannot be contained in the length of one line of the display device, the Help key must be used to display the entire literal.

The STOP literal statement is useful for special situations (a special tape or disk must be mounted, a specific daily code must be entered, and so forth) when operator intervention is needed during program execution. However, the ACCEPT and DISPLAY statements are preferred when operator intervention is needed.

When STOP RUN is specified, execution of the run unit is terminated, and control is returned to the system. If a STOP RUN statement appears in a sequence of imperative statements, it must be the last or only statement in the sequence. The STOP RUN statement closes all files.

For use of the STOP RUN statement in calling and called programs, see the table below.

Termination Statement	In a Main Program	In a Subprogram
STOP RUN	Return to calling program.	Return directly to the program that called the main program.

In each case above, the calling program could be the system. If it is, execution of the run unit ceases, and control transfers to the operating system.

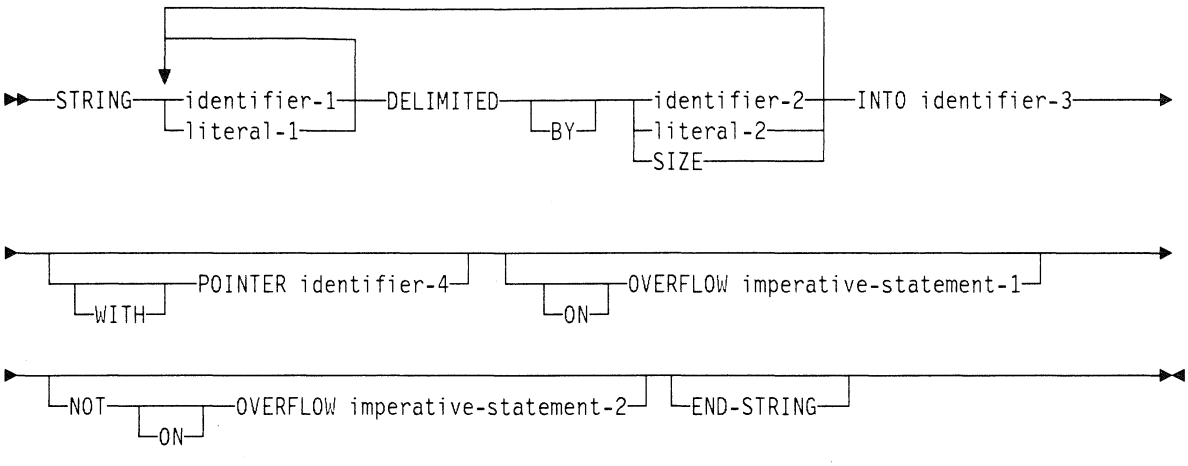
Also, if the main program is called by a program written in a language that does not follow COBOL linkage conventions, return will be to this calling program.

STRING Statement

The STRING statement strings together the partial or complete contents of two or more data items or literals into one single data item.

One STRING statement can be written instead of a series of MOVE statements.

Format



Note: All identifiers (except identifier-4, the POINTER item) must have USAGE DISPLAY, explicitly or implicitly.

identifier-1

Represents the **sending field(s)**.

When the sending field or any of the delimiters is an elementary numeric item, it must be described as an integer, and its PICTURE character-string must not contain the symbol P.

literal-1

Represents the **sending field(s)**.

All literals must be nonnumeric literals; each may be any figurative constant except the ALL literal. When a figurative constant is specified, it is considered a 1-character nonnumeric literal.

DELIMITED BY Phrase

The DELIMITED BY phrase sets the limits of the string.

identifier-2, literal-2

Are delimiters; that is, character(s) that delimit the data to be transferred.

If identifier-1 or identifier-2 occupies the same storage area as identifier-3 or identifier-4, undefined results will occur, even if the identifiers are defined by the same data description entry.

When a figurative constant is specified, it is considered a 1-character nonnumeric literal.

SIZE

Transfers the complete sending area.

INTO Phrase

identifier-3

Represents the **receiving field**.

It must not represent an edited data item and must not be described with the JUSTIFIED clause. It must not be reference modified.

If identifier-3 and identifier-4 occupy the same storage area, undefined results will occur, even if the identifiers are defined by the same data description entry.

POINTER Phrase

identifier-4

Represents the **pointer field**, which points to a character position in the receiving field.

It must be an elementary integer data item large enough to contain a value equal to the length of the receiving area plus 1. The pointer field must not contain the symbol P in its PICTURE character-string.

ON OVERFLOW Phrases

imperative-statement-1

Executed when the pointer value (explicit or implicit):

- Is less than 1
- Exceeds a value equal to the length of the receiving field.

When either of the above conditions occurs, an overflow condition exists, and no more data is transferred. Then the STRING operation is terminated, 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 that 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 STRING statement.

If at the time of execution of a STRING statement, conditions that would cause an overflow condition are not encountered, then after completion of the transfer of

data, the ON OVERFLOW phrase, if specified, is ignored. Control is then 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 that 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-2, control is transferred to the end of the STRING statement.

The ON OVERFLOW statement is not executed unless there was an attempt to move in one or more characters beyond the end of identifier-3.

END-STRING Phrase

This explicit scope terminator serves to delimit the scope of the STRING statement. END-STRING permits a conditional STRING statement to be nested in another conditional statement. END-STRING may also be used with an imperative STRING statement.

For more information, see "Delimited Scope Statements" on page 207.

Data Flow

When the STRING statement is executed, data is transferred from the sending fields to the receiving field. The order in which sending fields are processed is the order in which they are specified. The following rules apply:

- Characters from the sending fields are transferred to the receiving field, according to the rules for alphanumeric to alphanumeric elementary moves, except that no space filling is provided (see "MOVE Statement" on page 321).
- When DELIMITED BY identifier/literal is specified, the contents of each sending item are transferred, character-by-character, beginning with the leftmost character and continuing until either:
  - A delimiter for this sending field is reached (the delimiter itself is not transferred), or
  - The rightmost character of this sending field has been transferred.
- When DELIMITED BY SIZE identifier is specified, each entire sending field is transferred to the receiving field.
- When the receiving field is filled, or when all the sending fields have been processed, the operation is ended.
- When the POINTER phrase is specified, an explicit pointer field is available to the COBOL user to control placement of data in the receiving field. The user must set the explicit pointer's initial value, which must not be less than 1 and not more than the character count of the receiving field. (Note that the pointer field must be defined as a field large enough to contain a value equal to the length of the receiving field plus 1; this precludes arithmetic overflow when the system updates the pointer at the end of the transfer.)
- When the POINTER phrase is not specified, no pointer is available to the user. However, a conceptual implicit pointer with an initial value of 1 is used by the system.

- Conceptually, when the STRING statement is executed, the initial pointer value (explicit or implicit) is the first character position within the receiving field into which data is to be transferred. Beginning at that position, data is then positioned, character-by-character, from left to right. After each character is positioned, the explicit or implicit pointer is increased by 1. The value in the pointer field is changed only in this manner. At the end of processing, the pointer value always indicates a value equal to one character beyond the last character transferred into the receiving field.

Note: Any subscripting or reference modification is performed only once, at the beginning of the processing of the STRING statement. So if identifier-3 or identifier-4 is used as a subscript or reference modifier in the STRING statement, these values are determined at the beginning of the STRING statement, and are *not* affected by any results of the STRING statement.

After STRING statement execution is completed, only that part of the receiving field into which data was transferred is changed. The rest of the receiving field contains the data that was present before this execution of the STRING statement.

When the following STRING statement is executed, the results obtained will be like those illustrated in Figure 26.

```
STRING ID-1 ID-2 DELIMITED BY ID-3
       ID-4 ID-5 DELIMITED BY SIZE
  INTO ID-7 WITH POINTER ID-8
END-STRING
```

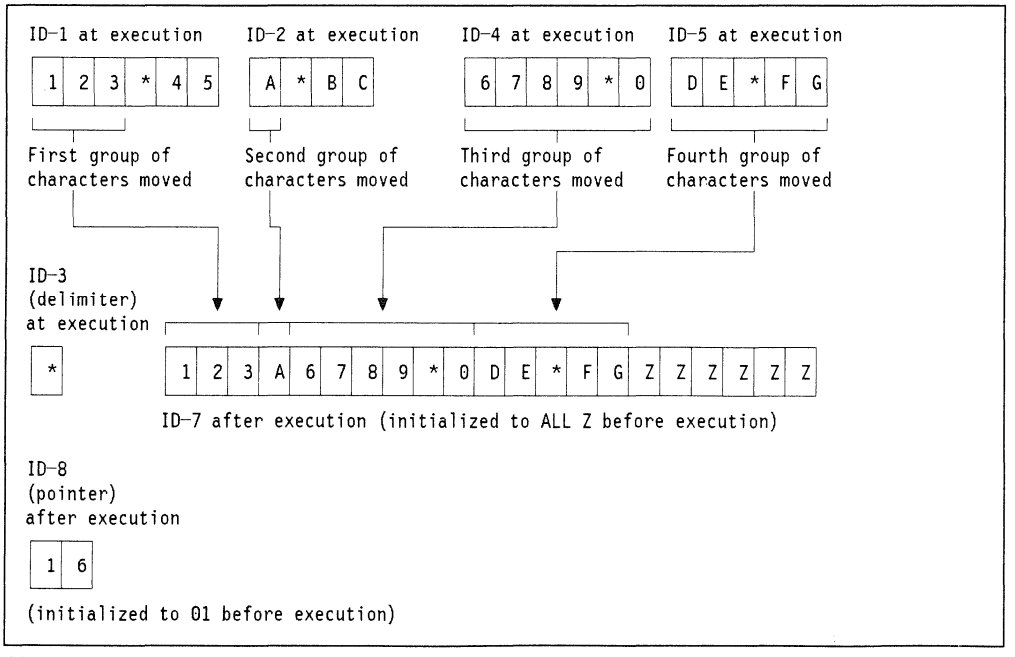


Figure 26. Results of STRING Statement Execution

STRING Statement Example

The following example illustrates some of the considerations that apply to the STRING statement.

In the Data Division, the programmer has defined the following fields:

```
01 RPT-LINE     PICTURE X(120).
01 LINE-POS     PICTURE 99.
01 LINE-NO      PICTURE 9(5) VALUE 1.
01 DEC-POINT    PICTURE X VALUE ".".
```

In the File Section, he or she has defined the following input record:

```
01 RCD-01.
   05 CUST-INFO.
      10 CUST-NAME  PICTURE X(15).
      10 CUST-ADDR  PICTURE X(34).
   05 BILL-INFO.
      10 INV-NO     PICTURE X(6).
      10 INV-AMT    PICTURE $$,$$$.99.
      10 AMT-PAID   PICTURE $$,$$$.99.
      10 DATE-PAID  PICTURE X(8).
      10 BAL-DUE    PICTURE $$,$$$.99.
      10 DATE-DUE   PICTURE X(8).
```

The programmer wants to construct an output line consisting of portions of the information from RCD-01. The line is to consist of a line number, customer name and address, invoice number, date due, and balance due, truncated to the dollar figure shown.

The record as read in contains the following information:

```
J.B.bSMITHbbbbb
444bSPRINGbST.,bCHICAGO,bILL.bbbbb
A14275
$4,736.85
$2,400.00
09/22/76
$2,336.85
10/22/76
```

In the Procedure Division, the programmer initializes RPT-LINE to SPACES and sets LINE-POS (which is to be used as the pointer field) to 4. Then he issues this STRING statement:

```
STRING LINE-NO SPACE
    CUST-INFO SPACE
    INV-NO SPACE
    DATE-DUE SPACE
  DELIMITED BY SIZE,
    BAL-DUE
  DELIMITED BY DEC-POINT
  INTO RPT-LINE
  WITH POINTER LINE-POS.
```

When the statement is executed, the following actions take place:

1. The field LINE-NO is moved into positions 4 through 8 of RPT-LINE.
2. A space is moved into position 9.

3. The group item CUST-INFO is moved into positions 10 through 58.
4. A space is moved into position 59.
5. INV-NO is moved into positions 60 through 65.
6. A space is moved into position 66.
7. DATE-DUE is moved into positions 67 through 74.
8. A space is moved into position 75.
9. The portion of BAL-DUE that precedes the decimal point is moved into positions 76 through 81.

After the STRING statement has been executed:

- RPT-LINE appears as shown in Figure 27.
- LINE-POS contains the value 82.

Programming Note: One STRING statement can be written instead of a series of MOVE statements.

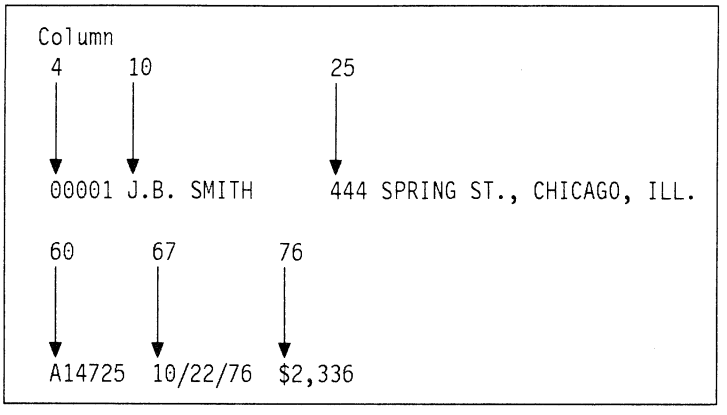
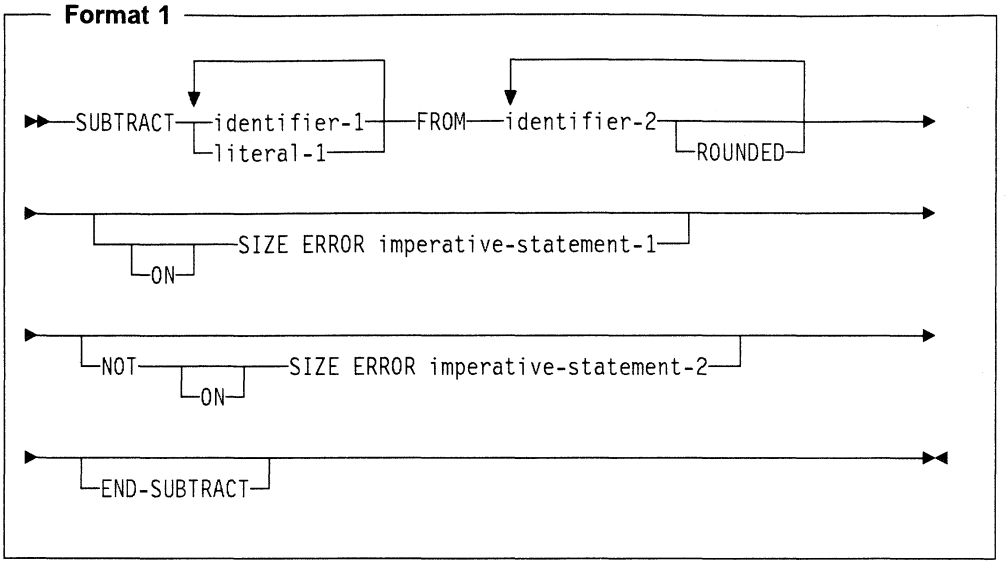


Figure 27. STRING Statement Example Output Data

SUBTRACT Statement

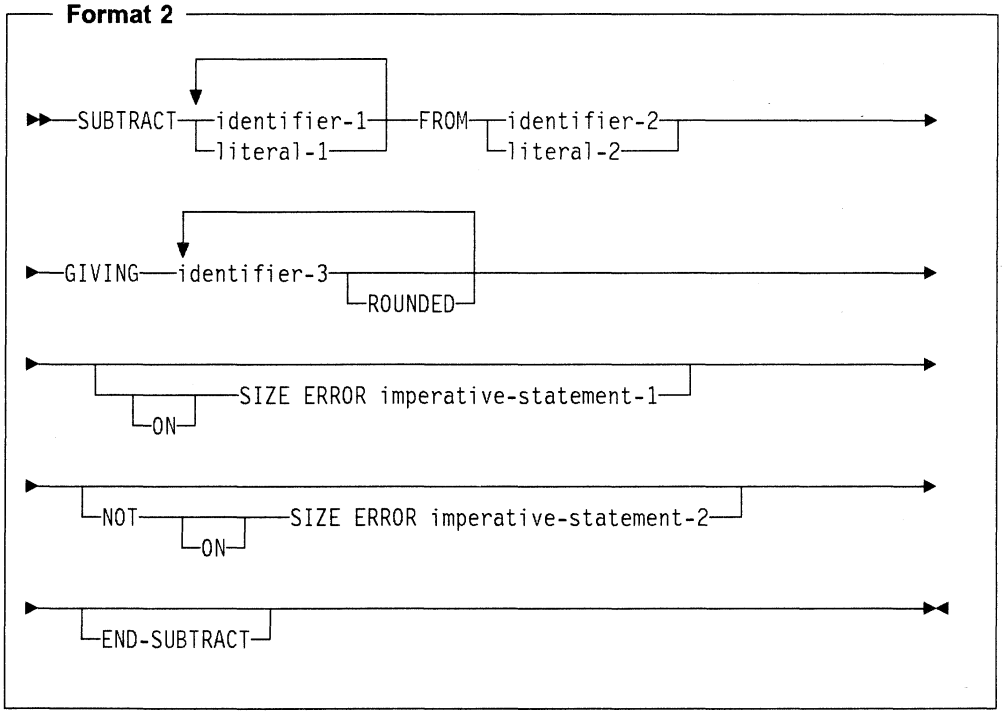
The SUBTRACT statement subtracts one numeric item, or the sum of two or more numeric items, from one or more numeric items, and stores the result.

Format 1



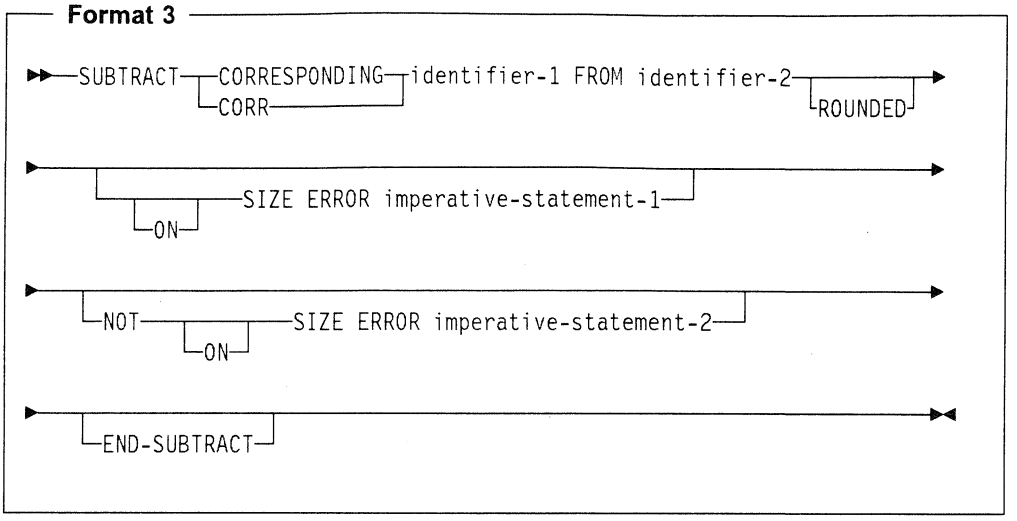
Identifiers and literals preceding the keyword FROM are added together, and this initial sum is subtracted from and stored immediately in identifier-2. The initial sum is also subtracted from each successive occurrence of identifier-2, in the left-to-right order in which identifier-2 is specified.

Format 2



All identifiers or literals preceding the key word FROM are added together and this sum is subtracted from identifier-2 or literal-2. The result of the subtraction is stored as the new value of each data item referenced by identifier-3.

Format 3



Elementary data items within identifier-1 are subtracted from, and the results are stored in, the corresponding elementary data items within identifier-2.

If the composite of operands is 18 digits or less, enough places are carried so that no significant digits are lost during execution.

IBM Extension

The composite of all operands in an arithmetic statement can have a maximum length of 30 digits.

End of IBM Extension

For all Formats:

identifier

In Format 1, must name an elementary numeric item.

In Format 2, must name an elementary numeric item, unless the identifier follows the word GIVING. Each identifier following the word GIVING must name a numeric or numeric-edited elementary item.

In Format 3, must name a group item.

literal

Must be a numeric literal.

ROUNDED Phrase

For information on the ROUNDED phrase, and for operand considerations, see "ROUNDED Phrase" on page 210.

SIZE ERROR Phrases

For information on the SIZE ERROR phrases, and for operand considerations, see "SIZE ERROR Phrases" on page 210.

CORRESPONDING Phrase (Format 3)

The CORRESPONDING phrase (CORR) allows operations to be performed on elementary numeric data-items of the same name if the group items to which they belong are specified.

The rules for use of the CORRESPONDING phrase with the SUBTRACT statement are the same as for the ADD statement. See "CORRESPONDING Phrase" on page 209 for more information.

END-SUBTRACT Phrase

This explicit scope terminator serves to delimit the scope of the SUBTRACT statement. END-SUBTRACT permits a conditional SUBTRACT statement to be nested in another conditional statement. END-SUBTRACT may also be used with an imperative SUBTRACT statement.

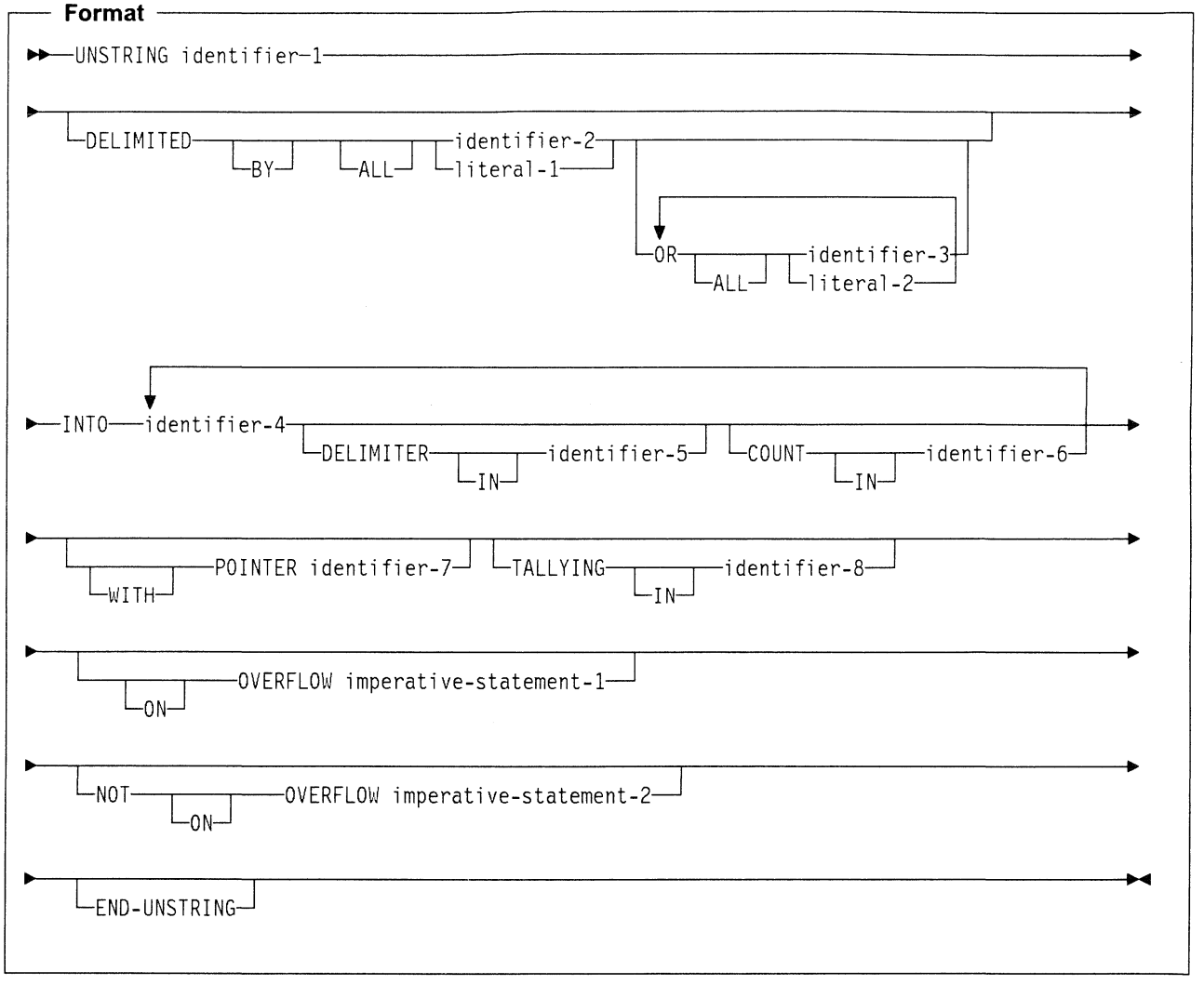
For more information, see "Delimited Scope Statements" on page 207.

UNSTRING Statement

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

One UNSTRING statement can be written instead of a series of MOVE statements.

Format



identifier-1

Represents the **sending field**.

It must be an alphanumeric data item; it cannot be reference modified. Data is transferred from this field to the receiving fields.

DELIMITED BY Phrase

This phrase specifies delimiters within the data that control the data transfer.

The delimiters are identifier-2, identifier-3, or their corresponding literals. Each identifier or literal specified represents one delimiter. Each must be an alphanumeric data item.

Unless the DELIMITED BY phrase is specified, the DELIMITER IN and COUNT IN phrases must not be specified.

identifier-2, identifier-3
Each represents one delimiter. Each must be an alphanumeric data item.

literal-1, literal-2
Each must be a nonnumeric literal; each may be any figurative constant except the ALL literal. When a figurative constant is specified, it is considered to be a 1-character nonnumeric literal.

ALL
One or more contiguous occurrences of any delimiters are treated as if they were only one occurrence, and this one occurrence is moved to the delimiter receiving field (if specified). The delimiting characters in the sending field are treated as an elementary alphanumeric item and are moved into the current delimiter receiving field, according to the rules of the MOVE statement.

When DELIMITED BY ALL is **not** specified, and two or more contiguous occurrences of any delimiter are encountered, the current data receiving field is filled with spaces or zeros, according to the description of the data receiving field.

If a delimiter contains two or more characters, it is recognized as a delimiter only if the delimiting characters are contiguous, and in the sequence specified in the sending field.

When two or more delimiters are specified, an OR condition exists, and each nonoverlapping occurrence of any one of the delimiters is recognized in the sending field in the sequence specified. For example, if DELIMITED BY "AB" or "BC" is specified, then an occurrence of either AB or BC in the sending field is considered a delimiter; an occurrence of ABC is considered an occurrence of AB. The data-count fields, the pointer field, and the field-count field must each be an integer item without the symbol P in the PICTURE character-string.

INTO Phrase

identifier-4
Represents the **data receiving fields**.

Each must have USAGE DISPLAY. These fields can be defined as:

- Alphabetic
- Alphanumeric
- Numeric (without the symbol P in the PICTURE string).

DELIMITER IN
Represents the **delimiter receiving fields**.

identifier-5
Must be alphanumeric.

Unless the DELIMITED BY phrase is specified, the DELIMITER IN phrase must not be specified. They must not be defined as alphanumeric edited or numeric edited items.

COUNT IN
Identifier-6 is the **data-count field** for each data transfer. Each field holds the count of examined characters in the sending field, terminated by the delim-

iters or the end of the sending field, for the move to this receiving field; the delimiters are not included in this count.

The COUNT IN phrase must not be specified unless the DELIMITED BY phrase is specified.

POINTER Phrase

identifier-7

Contains a value that indicates a relative position in the sending field. When this phrase is specified, the user must initialize this field before execution of the UNSTRING statement is begun.

TALLYING IN Phrase

identifier-8

The **field-count field** is increased by the number of data receiving fields acted upon in this execution of the UNSTRING statement. When this phrase is specified, the user must initialize this field before execution of the UNSTRING statement is begun.

ON OVERFLOW Phrases

imperative-statement-1

Executed when:

- The pointer value (explicit or implicit) is less than 1
- The pointer value (explicit or implicit) exceeds a value equal to the length of the sending field
- All data receiving fields have been acted upon, and the sending field still contains unexamined characters.

When any of the above conditions occurs, an overflow condition exists, and no more data is transferred. Then 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 that 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 UNSTRING statement.

If at the time of execution of an UNSTRING statement, conditions that would cause an overflow condition are not encountered, then after completion of the transfer of data, the ON OVERFLOW phrase, if specified, is ignored. Control is then 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 that 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-2, control is transferred to the end of the UNSTRING statement.

END-UNSTRING Phrase

This explicit scope terminator serves to delimit the scope of the UNSTRING statement. END-UNSTRING permits a conditional UNSTRING statement to be nested in another conditional statement. END-UNSTRING may also be used with an imperative UNSTRING statement.

For more information, see "Delimited Scope Statements" on page 207.

Data Flow

When the UNSTRING statement is initiated, data is transferred from the sending field to the current data receiving field, according to the following rules (the current data receiving field is identifier-4):

1. If the POINTER phrase is not specified, the sending field character-string is examined, beginning with the leftmost character. If the POINTER phrase is specified, the field is examined, beginning at the relative character position specified by the value in the pointer field.
2. If the DELIMITED BY phrase is specified, the examination proceeds from left to right, character-by-character, until a delimiter is encountered. If the end of the sending field is reached before a delimiter is found, the examination ends with the last character in the sending field. If there are more receiving fields, the next one is selected, otherwise, an overflow condition occurs.
3. If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current data receiving field, which depends on its data category:
   a. If the receiving field is alphanumeric or alphabetic, the number of characters examined is equal to the number of characters in the current receiving field.
   b. If the receiving field is numeric, the number of characters examined is equal to the number of characters in the integer portion of the current receiving field.
   c. If the receiving field is described with the SIGN IS SEPARATE clause, the number of characters examined is one less than the size of the current receiving field.
   d. If the receiving field is described as a variable-length data item, the number of characters examined is determined by the size of the current receiving field at the beginning of the UNSTRING operation.
4. The examined characters (excluding any delimiter characters) are treated as an alphanumeric elementary item, and are moved into the current data receiving field, according to the rules for the MOVE statement (see "MOVE Statement" on page 321).
5. If the DELIMITER IN phrase is specified, the delimiting characters in the sending field are treated as an elementary alphanumeric item and are moved to the current delimiter receiving field, according to the rules for the MOVE statement. If the delimiting condition is the end of the sending field, the current delimiter receiving field is filled with spaces.
6. If the COUNT IN phrase is specified, a value equal to the number of examined characters (excluding any delimiters) is moved into the data count field, according to the rules for an elementary move.
7. If the DELIMITED BY phrase is specified, the sending field is further examined, beginning with the first character to the right of the delimiter.

8. If the DELIMITED BY phrase is not specified, the sending field is further examined, beginning with the first character to the right of the last character examined.
9. For each succeeding data receiving field, the preceding procedure is repeated either until all the characters in the sending field have been transferred, or until there are no more unfilled data receiving fields.
10. When the POINTER phrase is specified, the value of the pointer field behaves as if it were increased by 1 for each examined character in the sending field. When this execution of the UNSTRING statement is completed, the pointer field contains a value equal to its initial value, plus the number of characters examined in the sending field.
11. When the TALLYING phrase is specified, then, when this execution of the UNSTRING statement is completed, the field-count field contains a value equal to the initial value, plus the number of data receiving areas acted upon.

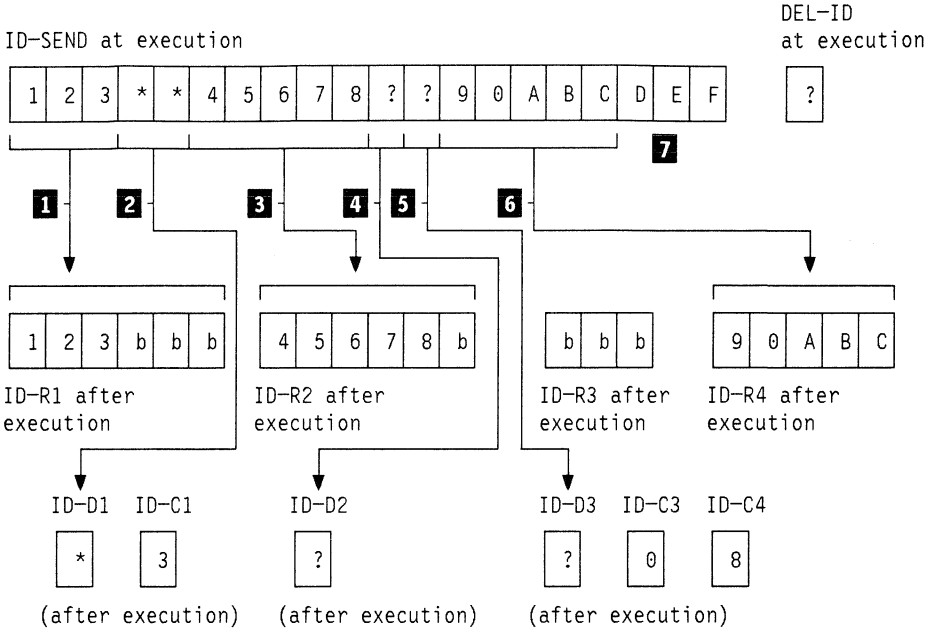
Note: All subscripting and reference modification is performed only once, at the beginning of the execution of the UNSTRING statement.

If any of the UNSTRING statement identifiers are subscripted or indexed, the subscripts and indexes are evaluated as follows:

- Any subscripting or indexing associated with the sending field, the pointer field, or the field-count field is evaluated only once, immediately before any data is transferred to any of the receivers.
- Any subscripting or indexing associated with the delimiters, the data and delimiter receiving fields, or the data-count fields, is evaluated immediately before the transfer of data into the affected data item.

Figure 28 on page 428 illustrates the rules of execution for the UNSTRING statement.

```
UNSTRING ID-SEND DELIMITED BY DEL-ID OR ALL "*"
    INTO ID-R1 DELIMITER IN ID-D1 COUNT IN ID-C1      (All the data
         ID-R2 DELIMITER IN ID-D2                     receiving fields
         ID-R3 DELIMITER IN ID-D3 COUNT IN ID-C3      are defined as
         ID-R4 COUNT IN ID-C4                         alphanumeric)
    WITH POINTER ID-P
    TALLYING IN ID-T
    ON OVERFLOW GO TO OFLOW-EXIT.
```



ID-P (pointer)

2	1

ID-T (tallying field)

0	5

(after execution—both initialized to 01 before execution)

The order of execution is:

1. 3 characters are placed in ID-R1.
2. Because ALL * is specified, all consecutive asterisks are processed, but only one asterisk is placed in ID-D1.
3. 5 characters are placed in ID-R2.
4. A ? is placed in ID-D2. The current receiving field is now ID-R3.
5. A ? is placed in ID-D3; ID-R3 is filled with spaces; no characters are transferred, so 0 is placed in ID-C3.
6. No delimiter is encountered before 5 characters fill ID-R4; 8 is placed in ID-C4, representing the number of characters examined since the last delimiter.
7. ID-P is updated to 21, the total length of the sending field + 1; ID-T is updated to 5, the number of fields acted upon + 1. Since there are no unexamined characters in the ID-SEND, the OVERFLOW EXIT is not taken.

Figure 28. Results of UNSTRING Statement Execution

UNSTRING Statement Example

The following example illustrates some of the considerations that apply to the UNSTRING statement.

In the Data Division, the user has defined the following input record to be acted upon by the UNSTRING statement:

```
01 INV-RCD.
   05 CONTROL-CHARS  PIC XX.
   05 ITEM-INDENT    PIC X(20).
   05 FILLER         PIC X.
   05 INV-CODE       PIC X(10).
   05 FILLER         PIC X.
   05 NO-UNITS       PIC 9(6).
   05 FILLER         PIC X.
   05 PRICE-PER-M    PIC 99999.
   05 FILLER         PIC X.
   05 RTL-AMT        PIC 9(6).99.
```

The next two records are defined as receiving fields for the UNSTRING statement. DISPLAY-REC is to be used for printed output. WORK-REC is to be used for further internal processing.

```
01 DISPLAY-REC
   05 INV-NO        PIC X(6).
   05 FILLER        PIC X VALUE SPACE
   05 ITEM-NAME     PIC X(20).
   05 FILLER        PIC X VALUE SPACE
   05 DISPLAY-DOLS  PIC 9(6).

01 WORK-REC
   05 M-UNITS       PIC 9(6).
   05 FIELD-A       PIC 9(6).
   05 WK-PRICE
      REDEFINES
      FIELD-A       PIC 9999V99.
   05 INV-CLASS     PIC X(3).
```

The user has also defined the following fields for use as control fields in the UNSTRING statement.

```
01 DBY-1          PIC X, VALUE IS ".".
01 CTR-1          PIC 99, VALUE IS ZERO.
01 CTR-2          PIC 99, VALUE IS ZERO.
01 CTR-3          PIC 99, VALUE IS ZERO.
01 CTR-4          PIC 99, VALUE IS ZERO.
01 DLTR-1         PIC X.
01 DLTR-2         PIC X.
01 CHAR-CT        PIC 99, VALUE IS 3.
01 FLDS-FILLED    PIC 99, VALUE IS ZERO.
```

In the Procedure Division, the user writes the following UNSTRING statement to move subfields of INV-RCD to the subfields of DISPLAY-REC and WORK-REC:

```
UNSTRING INV-RCD
 DELIMITED BY ALL SPACES
   OR "/"
   OR DBY-1
 INTO ITEM-NAME COUNT IN CTR-1,
 INV-NO DELIMITER IN DLTR-1
   COUNT IN CTR-2,
 INV-CLASS,
 M-UNITS COUNT IN CTR-3,
 FIELD-A,
 DISPLAY-DOLS DELIMITER IN DLTR-2
   COUNT IN CTR-4
 WITH POINTER CHAR-CT
 TALLYING IN FLDS-FILLED
 ON OVERFLOW
   GO TO UNSTRING-COMPLETE.
```

Before the UNSTRING statement is issued, the user places the value 3 in the CHAR-CT (the pointer item), so as not to work with the two control characters at the beginning of INV-RCD. In DBY-1, a period is placed for use as a delimiter, and in FLDS-FILLED (the tallying item) the value 0 is placed. The following data is then read into INV-RCD as shown in Figure 29.

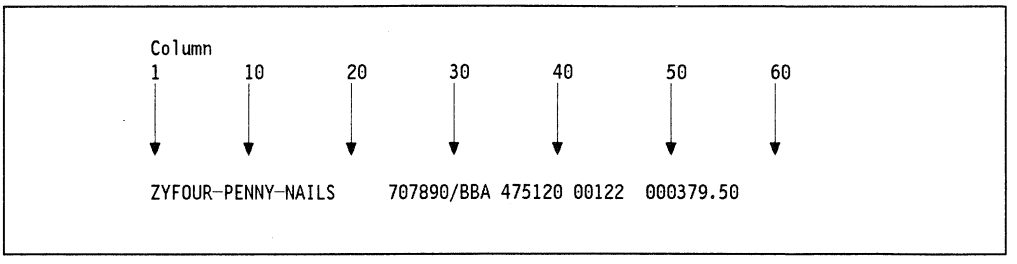


Figure 29. UNSTRING Statement Example — Input Data

When the UNSTRING statement is executed, the following actions take place:

1. Positions 3 through 18 (FOUR-PENNY-NAILS) of INV-RCD are placed in ITEM-NAME, left-justified within the area, and the unused character positions are padded with spaces. The value 16 is placed in CTR-1.
2. Because ALL SPACES is specified as a delimiter, the five contiguous SPACE characters are considered to be one occurrence of the delimiter.
3. Positions 24 through 29 (707890) are placed in INV-NO. The delimiter character / is placed in DLTR-1, and the value 6 is placed in CTR-2.
4. Positions 31 through 33 are placed in INV-CLASS. The delimiter is a SPACE, but because no field has been defined as a receiving area for delimiters, the SPACE is merely bypassed.
5. Positions 35 through 40 (475120) are examined and are placed in M-UNITS. The delimiter is a SPACE, but because no receiving field has been defined as a receiving area for delimiters, the SPACE is bypassed. The value 6 is placed in CTR-3.
6. Positions 42 through 46 (00122) are placed in FIELD-A and right-justified within the area. The high-order digit position is filled with a 0 (zero). The

delimiter is a SPACE, but because no field has been defined as a receiving area for delimiters, the SPACE is bypassed.

7. Positions 48 through 53 (000379) are placed in DISPLAY-DOLS. The period delimiter character is placed in DLTR-2, and the value 6 is placed in CTR-4.
8. Because all receiving fields have been acted upon and two characters of data in INV-RCD have not been examined, the ON OVERFLOW exit is taken, and execution of the UNSTRING statement is completed.

At the end of execution of the UNSTRING statement, DISPLAY-REC contains the following data:

```
707890 FOUR-PENNY-NAILS   000379
```

WORK-REC contains the following data:

```
475120000122BBA
```

CHAR-CT (the pointer field) contains the value 55, and FLD-FILLED (the tallying field) contains the value 6.

Programming Note: One UNSTRING statement can be written instead of a series of MOVE statements.

WRITE Statement

The WRITE statement releases a logical record for an output or input/output file.

When the WRITE statement is executed, the associated indexed or relative file must be open in OUTPUT, I-O, or EXTEND mode. The associated sequential file must be open in OUTPUT or EXTEND mode.

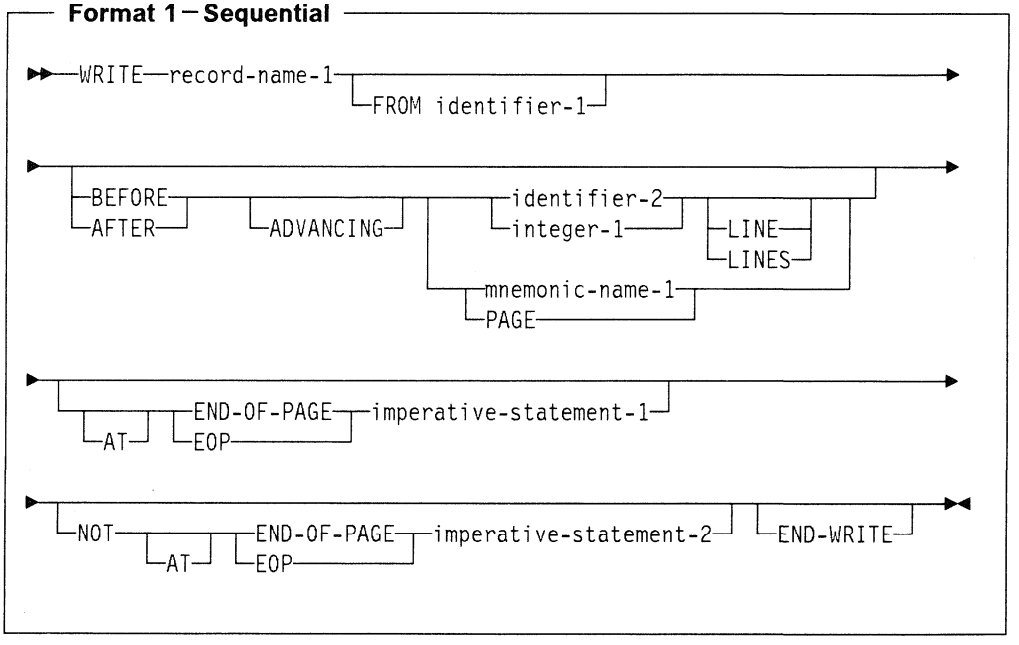
Sequential Files

Sequential files are sequentially organized. The ADVANCING and END-OF-PAGE phrases control the vertical positioning of each line on a printed page.

Note: The ADVANCING PAGE and END-OF-PAGE phrases must not both be specified in a single WRITE statement.

If the printed page is held on an intermediate device (a disk, for example), the format may appear different than the expected output when it is edited or browsed.

Format 1—Sequential



record-name-1
Must be defined in a Data Division FD entry. Record-name-1 may be qualified. It must not be associated with a sort or merge file.

FROM
When FROM is specified, the result is the same as:

```
MOVE identifier-1 TO record-name-1
WRITE record-name-1
```

After the WRITE statement is executed, the information is still available in identifier-1, even though it may not be in record-name-1. (See "INTO/FROM Identifier Phrase" on page 215.)

identifier-1
Must be an alphanumeric or numeric edited data item. Data is transferred from this field to the receiving fields.

identifier-2
Must be an integer data item.

The maximum record size for the file is established at the time the file is created, and cannot subsequently be changed.

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

After the WRITE statement is executed, the logical record is no longer available in record-name-1, unless:

- The associated file is named in a SAME RECORD AREA clause (in which case, the record is also available as a record of the other files named in the SAME RECORD AREA clause), or
- The WRITE statement is unsuccessful because of a boundary violation.

In either of these two cases, the logical record is still available in record-name-1.

The file position indicator is not affected by execution of the WRITE statement.

The number of character positions required to store the record in a file may or may not be the same as the number of character positions defined by the logical description of that record in the COBOL program. (See "PICTURE Clause Editing" on page 153 and "USAGE Clause" on page 171.)

If the FILE STATUS clause is specified in the File-Control entry, the associated status key is updated when the WRITE statement is executed, whether or not execution is successful.

The WRITE statement may not be executed for a sequential file opened in I-O mode.

ADVANCING Phrase

The ADVANCING phrase controls positioning of the output record on the page.

When this phrase is specified, the following rules apply:

1. When BEFORE ADVANCING is specified, the line is printed before the page is advanced.
2. When AFTER ADVANCING is specified, the page is advanced before the line is printed.
3. When identifier-2 is specified, the page is advanced the number of lines equal to the current value in identifier-2. Identifier-2 must name an elementary integer data item.
4. When integer is specified, the page is advanced the number of lines equal to the value of integer.
5. Integer or the value in identifier-2 may be zero.

6. When mnemonic-name is specified, a system-specific action takes place. For more information on acceptable values for mnemonic-name, see "SPECIAL-NAMES Paragraph" on page 57. Mnemonic-name must be equated with environment-name-1 in the SPECIAL-NAMES paragraph (valid environment-names are listed in Table 3 on page 59).

   Note: This phrase is not valid if a LINAGE clause is specified in the FD entry for this file.

7. When PAGE is specified, the record is printed on the logical page BEFORE or AFTER (depending on the phrase used) the device is positioned to the next logical page. If PAGE has no meaning for the device used, then BEFORE or AFTER (depending on the phrase specified) ADVANCING 1 LINE is provided.

   If the FD entry contains a LINAGE clause, the repositioning is to the first printable line of the next page, as specified in that clause. If the LINAGE clause is omitted, the repositioning is to line 1 of the next succeeding page.

   LINAGE-COUNTER Rules: If the LINAGE clause is specified for this file, the associated LINAGE-COUNTER special register is modified during the execution of the WRITE statement, according to the following rules:

   1. If ADVANCING PAGE is specified, LINAGE-COUNTER is reset to 1.
   2. If ADVANCING identifier-2 or integer is specified, LINAGE-COUNTER is increased by the value in identifier-2 or integer.
   3. If the ADVANCING phrase is omitted, LINAGE-COUNTER is increased by 1.
   4. When the device is repositioned to the first available line of a new page, LINAGE-COUNTER is reset to 1.

When this phrase is omitted, automatic line advancing is provided, as if the user had written AFTER ADVANCING 1 LINE.

END-OF-PAGE Phrase

When END-OF-PAGE is specified, and the logical end of the printed page is reached during execution of the WRITE statement, the END-OF-PAGE imperative-statement is executed. When the END-OF-PAGE phrase is specified, the FD entry for this file must contain a LINAGE clause.

The logical end of the printed page is specified in the associated LINAGE clause.

An END-OF-PAGE condition is reached when execution of a WRITE END-OF-PAGE statement causes printing or spacing within the footing area of a page body. This occurs when execution of such a WRITE statement causes the value in the LINAGE-COUNTER special register to equal or exceed the value specified in the WITH FOOTING phrase of the LINAGE clause. The WRITE statement is executed, and then the END-OF-PAGE imperative-statement is executed.

An automatic page overflow condition is reached whenever the execution of any given WRITE statement (with or without the END-OF-PAGE phrase) cannot be completely executed within the current page body. This occurs when a WRITE statement, if executed, would cause the value in the LINAGE-COUNTER to exceed the number of lines for the page body specified in the LINAGE clause. In this case, the line is printed BEFORE or AFTER (depending on the option specified) the device is repositioned to the first printable line on the next logical page, as specified in the LINAGE clause. If the END-OF-PAGE phrase is specified, the END-OF-PAGE imperative-statement is then executed.

If the WITH FOOTING phrase of the LINAGE clause is not specified, the automatic page overflow condition exists because no end-of-page condition (as distinct from the page overflow condition) can be detected.

If the WITH FOOTING phrase is specified, but the execution of a given WRITE statement would cause the LINAGE-COUNTER to exceed both the footing value and the page body value specified in the LINAGE clause, then both the end-of-page condition and the automatic page overflow condition occur simultaneously.

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

Note: The phrases ADVANCING PAGE and END-OF-PAGE must not both be specified in a single WRITE statement.

END-WRITE Phrase

This explicit scope terminator serves to delimit the scope of the WRITE statement. END-WRITE permits a conditional WRITE statement to be nested in another conditional statement. END-WRITE may also be used with an imperative WRITE statement.

For more information, see "Delimited Scope Statements" on page 207.

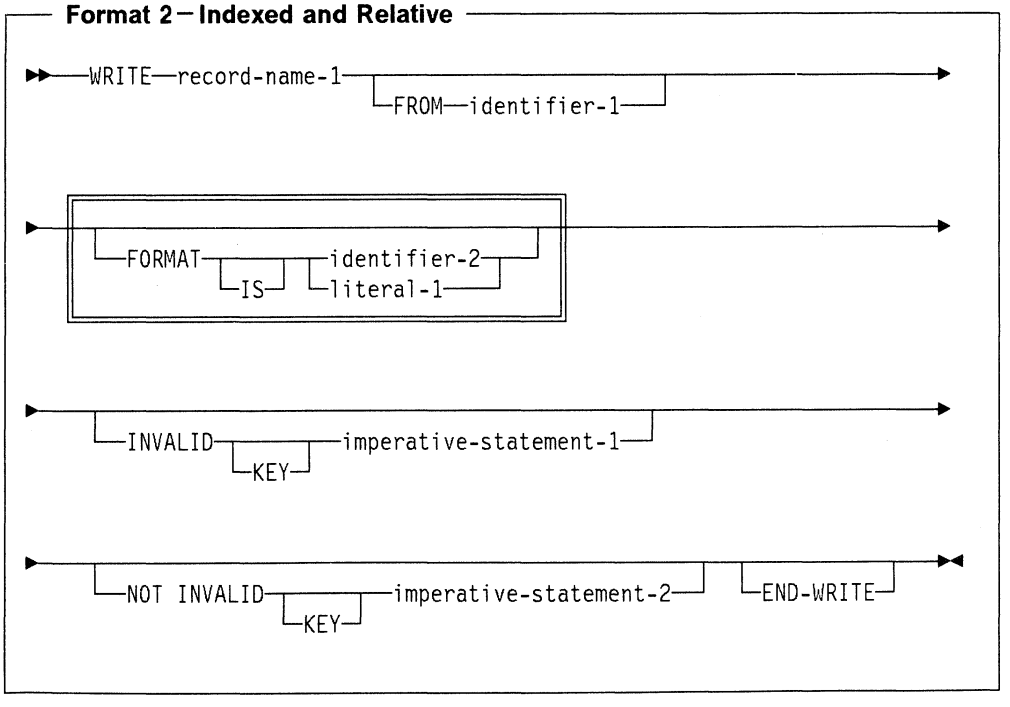
Multivolume Files

When end-of-volume is recognized for a multivolume OUTPUT file (tape or sequential direct-access file), the WRITE statement performs the following operations:

- The standard ending volume label procedure
- A volume switch
- The standard beginning volume label procedure.

Indexed and Relative Files

Format 2—Indexed and Relative



record-name-1
Must be defined in a Data Division FD entry. Record-name-1 may be qualified. It must not be associated with a sort or merge file.

FROM
When FROM is specified, the result is the same as:

```
MOVE identifier-1 TO record-name-1
WRITE record-name-1
```

After the WRITE statement is executed, the information is still available in identifier-1, even though it may not be in record-name-1. (See "INTO/FROM Identifier Phrase" on page 215.)

identifier-1
Must be an alphanumeric or numeric-edited data item. Data is transferred from this field to the receiving fields.

Record-name-1 and identifier-1 cannot both refer to the same storage area.

Before the WRITE statement is executed, you must set the prime record key (the RECORD KEY data item, as defined in the File-Control entry) to the desired value. (Note that RECORD KEY values must be unique within a file.)

When ACCESS IS SEQUENTIAL is specified in the File-Control entry, records must be released in ascending order of RECORD KEY values.

When ACCESS is RANDOM or ACCESS IS DYNAMIC is specified in the File-Control entry, records may be released in any programmer-specified order.

INVALID KEY Phrase

The INVALID KEY phrase must be specified if an explicit or implicit EXCEPTION/ERROR procedure is not specified for this file.

When an attempt is made to write beyond the externally defined boundaries of the file, WRITE statement execution is unsuccessful and an EXCEPTION/ERROR condition exists.

An invalid key condition is caused by any of the following:

- ACCESS SEQUENTIAL is specified and the file is opened OUTPUT, and the value of the prime record key is not greater than that of the previous record. Note that any signs on the record keys are ignored, even if the keys are defined as signed numeric in the DDS for the file.
- The file is opened OUTPUT or I-O and the value of the prime record key equals that of an already existing record.
- An attempt is made to write beyond the externally defined boundaries of the file.

When the invalid key condition is recognized, WRITE statement execution is unsuccessful, and the contents of the record are unaffected. Program execution proceeds according to the rules described under "INVALID KEY Condition" on page 215.

If the NOT INVALID KEY phrase is specified and a valid key condition exists at the end of the execution of the WRITE statement, control is passed to the imperative statement associated with this phrase.

END-WRITE Phrase

This explicit scope terminator serves to delimit the scope of the WRITE statement. END-WRITE permits a conditional WRITE statement to be nested in another conditional statement. END-WRITE may also be used with an imperative WRITE statement.

For more information, see "Delimited Scope Statements" on page 207.

Relative Files

record-name-1
Must be defined in a Data Division FD entry. Record-name-1 may be qualified.

The number of character positions in record-name-1 must equal the number of character positions in the record being replaced. It must not be associated with a sort or merge file.

FROM
When FROM is specified, the result is the same as:

```
MOVE identifier-1 TO record-name-1
WRITE record-name-1
```

After the WRITE statement is executed, the information is still available in identifier-1, even though it may not be in record-name-1. (See "INTO/FROM Identifier Phrase" on page 215.)

Record-name-1 and identifier-1 cannot both refer to the same storage area.

identifier-1
Must be an alphanumeric or numeric-edited data item. Data is transferred from this field to the receiving fields.

For OUTPUT files, the WRITE statement causes the following actions:

- If ACCESS IS SEQUENTIAL is specified:

  The first record released has relative record number 1, the second record released has relative record number 2, the third number 3, and so on.

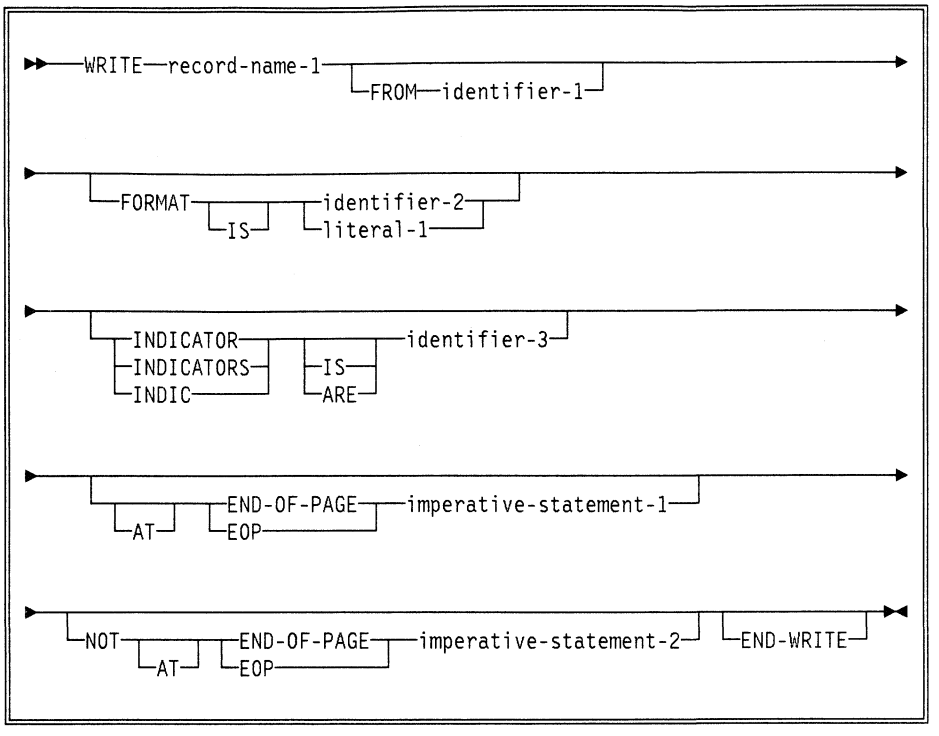
  If the RELATIVE KEY is specified in the File-Control entry, the relative record number of the record just released is placed in the RELATIVE KEY during execution of the WRITE statement.

- If ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified, the RELATIVE KEY must contain the desired relative record number for this record before the WRITE statement is issued. When the WRITE statement is executed, this record is placed at the specified relative record number position in the file.

For I-O files, either ACCESS IS RANDOM or ACCESS IS DYNAMIC must be specified; the WRITE statement inserts new records into the file. The RELATIVE KEY must contain the desired relative record number for this record before the WRITE statement is issued. When the WRITE statement is executed, this record is placed at the specified relative record number position in the file.

For a physical file that does not allow the DELETE operation on records (e.g. CRTPF with ALWDLT(*NO)), the update operation on records must be allowed (i.e. CRTPF with ALWUPD(*YES)).

Format 3 — FORMATFILE



WRITE Statement Considerations

The following tables illustrate organization, access, and device considerations for the WRITE statement. The letter codes used in the tables are defined in the section following the tables.

Table 52. Sequential Organization

ACCESS	SEQUENTIAL					
DEVICE	**PRINTER**	**TAPEFILE**	**DISKETTE**	**DISK**	**DATABASE**	**FORMATFILE**
WRITE Verb	Q,T	V,Q,S,T	V,Q,T	Q,S,T,Z	Q,S,T,Z	Q,T
FROM	O,B	O,B	O,B	O,B	O,B	O,B
INVALID KEY	—	—	—	—	—	—
NOT INVALID KEY	—	—	—	—	—	—
ADVANCING	O,D	—	—	—	—	—
AT END-OF-PAGE	O,E1,E3	—	—	—	—	O,E2,E3
NOT AT END-OF-PAGE	E3,E4	—	—	—	—	E3,E4
FORMAT	—	—	—	—	—	N,F
INDICATORS	—	—	—	—	—	I

Table 53. Relative Organization

Device	DISK			DATABASE		
Access	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**
WRITE Verb	Q,K,Z	P,Q,M,Z	P,M,Q,Z	Q,K,Z	P,M,Q,Z	P,M,Q,Z
FROM	O,B	O,B	O,B	O,B	O,B	O,B
INVALID KEY	O,J1,U1	O,J1,J2,U1	O,J1,J2,U1	O,J1,U1	O,J1,J2,U1	O,J1,J2,U1
NOT INVALID KEY	O,U2	O,U2	O,U2	O,U2	O,U2	O,U2
ADVANCING	—	—	—	—	—	—
AT END-OF-PAGE	—	—	—	—	—	—
NOT AT END-OF-PAGE	—	—	—	—	—	—
FORMAT	—	—	—	—	—	—
INDICATORS	—	—	—	—	—	—

Table 54. Indexed Organization

Device	DISK			DATABASE		
Access	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**	**SEQUENTIAL**	**RANDOM**	**DYNAMIC**
WRITE Verb	G,H1,Q,Z	G,H2,P,Q,Z	G,H2,P,Q,Z	G,H1,Q,Z	G,H2,P,Q,Z	G,H2,P,Q,Z
FROM	O,B	O,B	O,B	O,B	O,B	O,B
INVALID KEY	O,J1,J4,U1	O,J1,J3,U1	O,J1,J3,U1	O,J1,J4,U1	O,J1,J3,U1	O,J1,J3,U1
NOT INVALID KEY	O,U2	O,U2	O,U2	O,U2	O,U2	O,U2
ADVANCING	—	—	—	—	—	—
AT END-OF-PAGE	—	—	—	—	—	—
NOT AT END-OF-PAGE	—	—	—	—	—	—
FORMAT	—	—	—	N,F	N,F	N,F
INDICATORS	—	—	—	—	—	—

Letter Code **Meaning**

— Combination is not valid.

A When the KEY phrase is not specified, the file position indicator is set to the record in the file with a key (relative record number) equal to the RELATIVE KEY data item.

B The FROM identifier phrase makes a WRITE statement equivalent to:

```
MOVE identifier TO record-name
WRITE record-name.
```

After successful processing of the WRITE statement, the current record pointer is no longer available in record-name, but is still available in identifier. Record-name and identifier cannot both refer to the same storage area.

D When not specified, a default of AFTER ADVANCING 1 LINE is used. When specified, the following rules apply:

- When BEFORE ADVANCING is specified, the line is printed before the page advances.

- When AFTER ADVANCING is specified, the page advances before the line is printed.
- When identifier-2 is specified, the page advances the number of lines equal to the current value in identifier-2. Identifier-2 must name an elementary integer data item. Identifier-2 can be zero.
- When integer is specified, the page advances the number of lines equal to the value of integer. Integer or the value in identifier-2 can be zero.
- When a mnemonic-name is specified, a page eject or space suppression occurs. The mnemonic-name must be equated with function-name-1 in the SPECIAL-NAMES paragraph. This phrase is not valid if a LINAGE clause is specified in the FD entry for this file.
- When PAGE is specified, the record is printed on the logical page BEFORE or AFTER (depending on the phrase specified) the device is positioned to the next logical page. If PAGE has no meaning for the device used, BEFORE or AFTER ADVANCING 1 LINE is provided (depending on the phrase specified).

  If the FD entry contains a LINAGE clause, the device is positioned to the first printable line of the next page, as specified in that clause. If the LINAGE clause is omitted, the device is positioned to line 1 of the next page.

  If the LINAGE clause is specified for this file, the associated LINAGE-COUNTER special register is modified during the processing of the WRITE statement, according to the following rules:

  - If ADVANCING PAGE is specified, LINAGE-COUNTER is reset to 1.
  - If ADVANCING identifier-2 or integer is specified, LINAGE-COUNTER is incremented by the value of identifier-2 or integer.
  - If the ADVANCING phrase is omitted, LINAGE-COUNTER is incremented by 1.
  - When the device is repositioned to the first printable line of a new page, LINAGE-COUNTER is reset to 1.

When this phrase is omitted, automatic line advancing is provided, as if the user had written AFTER ADVANCING 1 LINE.

E1 The keywords END-OF-PAGE and EOP are equivalent. When the END-OF-PAGE phrase is specified, the FD entry for this file must contain a LINAGE clause. When END-OF-PAGE is specified, and an END-OF-PAGE condition exists after the processing of the WRITE statement, the END-OF-PAGE imperative-statement is processed. The logical end of the printed page is specified in the LINAGE clause associated with record-name.

An END-OF-PAGE condition for a printer file is reached when the processing of a WRITE statement for that file causes printing or spacing within the footing area of a page body. This occurs when the processing of such a WRITE statement causes the value in the LINAGE-COUNTER to equal or exceed the value specified in the WITH FOOTING phrase of the LINAGE clause. The WRITE statement is processed, and then the END-OF-PAGE imperative statement is processed, if coded.

An automatic page overflow condition is reached whenever the processing of any WRITE statement with or without the END-OF-PAGE phrase cannot be completely processed within the current page body. This occurs when a processed WRITE statement would cause the value in the LINAGE-COUNTER to exceed the number of lines for the page body specified in the LINAGE clause. In this case, the line is printed before or after (depending on the option specified) the device is repositioned to the first printable line on the next logical page, as specified in the LINAGE clause.

If the END-OF-PAGE phrase is specified, the END-OF-PAGE imperative-statement is then processed. The END-OF-PAGE condition and automatic page overflow condition occur simultaneously in the following cases:

- When the WITH FOOTING phrase of the LINAGE clause is not specified. This results in no distinction between the END-OF-PAGE condition and the page overflow condition. No footing information can be printed at the bottom of a logical page when the FOOTING phrase is not specified.
- When the WITH FOOTING phrase is specified, but the processing of a WRITE statement would cause the LINAGE-COUNTER to exceed both the footing value and the page body value specified in the LINAGE clause.

The keywords END-OF-PAGE and EOP are equivalent.

Note: The phrases ADVANCING PAGE and END-OF-PAGE must not both be specified in a single WRITE statement.

E2 The keywords END-OF-PAGE and EOP are equivalent. When the END-OF-PAGE phrase is specified, and an EOP condition exists after the processing of the WRITE statement for the FORMATFILE file, the END-OF-PAGE imperative statement is processed. An EOP condition for a FORMATFILE file occurs when the logical end of page is reached during the processing of a WRITE statement for that file. The logical end of the printed page is specified in the overflow line number parameter of the CRTPRTF command or the OVRPRTF command.

E3 Once the END-OF-PAGE condition becomes true, an EOP imperative statement must be processed to make it false again.

E4 If an END-OF-PAGE condition does not exist after the processing of a WRITE statement with the NOT AT END-OF-PAGE phrase, control transfers to the imperative statement associated with that phrase.

IBM Extension

F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format to operate on.

If an identifier is specified, it must be a character-string of ten characters or less, and it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

If a literal is specified, it must be an uppercase character-string of ten characters or less.

A value of all blanks is treated as though the FORMAT phrase were not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is invoked, if applicable for the file.

End of IBM Extension

G When the WRITE statement is processed, the system releases the record. Before the WRITE statement is processed, the user must set the key fields in the record area to the desired value.

IBM Extension

If the DUPLICATES phrase is specified, record key values for a format need not be unique (see RECORD KEY information on page 86). In this case, the system stores the records so that later sequential access to the records allows retrieval in the order specified in DDS.

End of IBM Extension

H1 Records must be released in ascending RECORD KEY value sequence.

Note: The records must be released in ascending key sequence even though the file can be ordered in descending key sequence by a DDS option.

H2 Records can be released in any user-specified order.

I Refer to the text under "INDICATORS" in the *COBOL/400 User's Guide*.

J1 An INVALID KEY condition exists when an attempt is made to write beyond the externally defined boundaries of the file.

J2 An INVALID KEY condition exists when RELATIVE KEY specifies a record that already contains data.

J3 An INVALID KEY condition exists when the value of the key field in the record area equals that of an already existing record and DUPLICATES are not allowed.

J4 An INVALID KEY condition exists when the value of the key field in the record area is not greater than that for the previous record.

IBM Extension

For a file that allows duplicate keys, the INVALID KEY condition exists only if the value of the record key is less than that for the previous record.

End of IBM Extension

K The first record released has relative record number 1, the second has number 2, the third has number 3, and so on. If the RELATIVE KEY is specified in the file-control entry, the relative record number of the record just released is placed in the RELATIVE KEY during processing of the WRITE statement.

M The RELATIVE KEY must contain the desired relative record number for this record before the WRITE statement is issued. When the WRITE statement is processed, this record is placed at the specified relative record number position in the file, if this position is vacant.

N Required if there is more than one record format for the file.

O You can specify this combination.

P Allowed when the file is opened for I-O.

Q Allowed when the file is opened for OUTPUT.

S Allowed when the file is opened for EXTEND.

T When an attempt is made to write beyond the externally defined boundaries of the file, the processing of the WRITE statement is unsuccessful and an EXCEPTION/ERROR condition exists. The contents of record-name are unaffected. Processing then follows the rules for error handling as described under "USE Statement Programming Notes" on page 476.

U1 The INVALID KEY phrase must be specified for files in which an applicable USE procedure is not specified.

For information about INVALID KEY phrase processing, see "INVALID KEY Condition" on page 215.

U2 After the successful completion of a WRITE statement with the NOT INVALID KEY phrase, control transfers to the imperative statement associated with the phrase.

V When end-of-volume is recognized for a multivolume OUTPUT file, the WRITE statement processes the following operations in the following order:

1. The standard ending volume label procedure is run.
2. A volume switch occurs.
3. The standard beginning volume label procedure is run.

No indication that an end-of-volume has occurred is returned to the program.

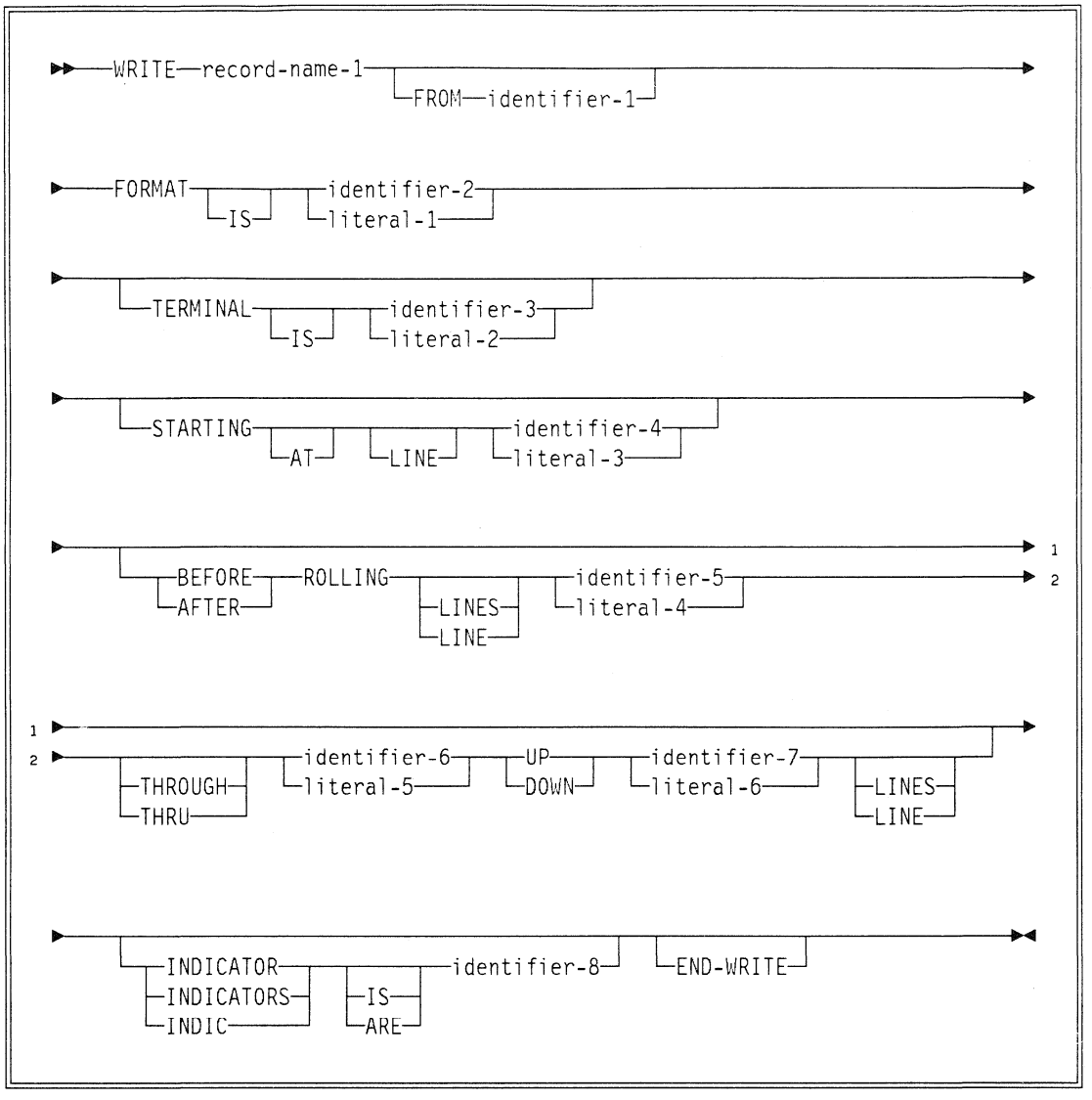
IBM Extension

Z The action of this statement can be inhibited at program run time by the INHWRT parameter of the OVRDBF CL command. When this parameter is specified, non-zero file status codes are not set for data dependent errors. Duplicate key and data conversion errors are examples of data dependent errors.

See the *CL Reference* for more information on this command.

End of IBM Extension

Format 4—TRANSACTION (Nonsubfile)



TERMINAL Phrase

The TERMINAL phrase specifies the program devices to which the output record is to be sent.

The contents of literal-2 or identifier-3 must be the name of a program device previously acquired, either implicitly or explicitly, by the file. Literal-2, if specified, must be nonnumeric and 10 characters or less in length. Identifier-3, if specified, must refer to an alphanumeric data item, 10 characters or less in length. A value of blanks is treated as if the TERMINAL phrase was omitted.

If only a single program device was acquired by the TRANSACTION file, the TERMINAL phrase can be omitted. That program device is always used for the WRITE.

If the TERMINAL phrase is omitted for a WRITE operation to a TRANSACTION file that has acquired multiple program devices, the default program device is used.

STARTING Phrase

The STARTING phrase specifies the starting line number for the record formats that use the variable starting line keyword. This phrase is only valid for display devices.

The actual line number on which a field begins can be determined from the following equation:

```
Actual-line = Start-line + DDS Start-line −1
```

Figure 30. Line Number Equation for the STARTING Phrase

Where:

Actual-line is the actual line number
Start-line is the starting line number specified in the program
DDS Start-line is the line number specified in positions 39 through 41 of the Data Description Specifications form.

The write is successful if:

- The result of the above equation is positive and less than or equal to the number of lines on the work station screen.
- The value specified for the STARTING phrase is 0. In this case, a value of 1 is assumed.

The write is unsuccessful and the program terminates if:

- The result of the above equation is greater than the number of lines on the work station screen.
- The value specified for the STARTING phrase is negative.

If the value specified for the STARTING phrase is within the screen area, any fields outside of the screen area are ignored.

Literal-3 of the STARTING phrase must be a numeric literal. Identifier-4 must be an elementary numeric item.

To use the STARTING phrase, the DDS record level keyword SLNO(*VAR) must be specified for the format being written. If the record format does not specify this keyword, the STARTING phrase is ignored at execution time.

The DDS keyword CLRL also affects the STARTING phrase. CLRL controls how much of the screen is cleared when the WRITE statement is executed.

See the *DDS Reference* for further information on SLNO(*VAR) and CLRL.

ROLLING Phrase

The ROLLING phrase allows you to move lines displayed on the work station screen. All or some of the lines on the screen can be rolled up or down. The lines vacated by the rolled lines are cleared, and can have another screen format written into them. This phrase is only valid for display devices.

ROLLING is specified in the WRITE statement that is writing a new format to the work station screen. You must specify whether the write is before or after the roll, the range of lines you want to roll, how many lines you want to roll these lines, and whether the roll operation is up or down.

After lines are rolled, the fields on these lines retain their DDS display attributes, for example, underlining, but lose their DDS usage attributes, for example, input-capability. Fields on lines that are written and then rolled (BEFORE ROLLING phrase) also lose their usage attributes.

If any part of a format is rolled, the entire format loses its usage attributes. If more than one format exists, only the rolled formats lose their usage attributes.

When you specify the ROLLING phrase, the following general rules apply.

- The DDS record level keyword ALWROL must be specified for every record format written in a WRITE statement containing the ROLLING phrase.
- Other DDS keywords mutually exclusive with the ALWROL keyword must not be used; see the *DDS Reference*.
- Either of the DDS keywords, CLRL or OVERLAY, must be specified for a record format that is to be written and rolled to prevent the display screen from being cleared when that record format is written. See the *DDS Reference*.
- All the identifiers and literals must represent positive integer values.
- The roll starting line number (identifier-5 or literal-4) must not exceed the ending line number (identifier-6 or literal-5).
- The contents of lines that are rolled outside of the window specified by the starting and ending line numbers disappear.

Figure 31 shows an example of rolling. An initial screen format, FMT1 is written on the work station screen. The program processes this screen format and is now ready to write the next screen format, FMT2, to the work station screen. Part of FMT1 is rolled down 2 lines before FMT2 is written to the work station screen.

Execution of the following WRITE statement causes part of FMT1 to be rolled down 2 lines, and FMT2 to be written to the work station screen:

```
WRITE SCREENREC FORMAT "FMT2"
 AFTER ROLLING LINES 14 THROUGH 20
 DOWN 2 LINES
```

When this WRITE statement is executed, the following steps occur:

1. The contents of lines 14 through 20 are rolled down 2 lines.
   a. The contents of lines 14 through 18 now appear on lines 16 through 20.
   b. The contents of lines 14 and 15 are vacated and cleared.
   c. The contents of lines 19 and 20 are rolled outside the window and disappear.
2. After the rolling operation takes place, FMT2 is written to the work station screen.
   a. Part of FMT2 is written to the area vacated by the roll operation.
   b. Part of FMT2 is written over the data left from FMT1.
3. When the contents of the work station screen are returned to the program by a READ statement, only the input capable fields of FMT2 are returned.

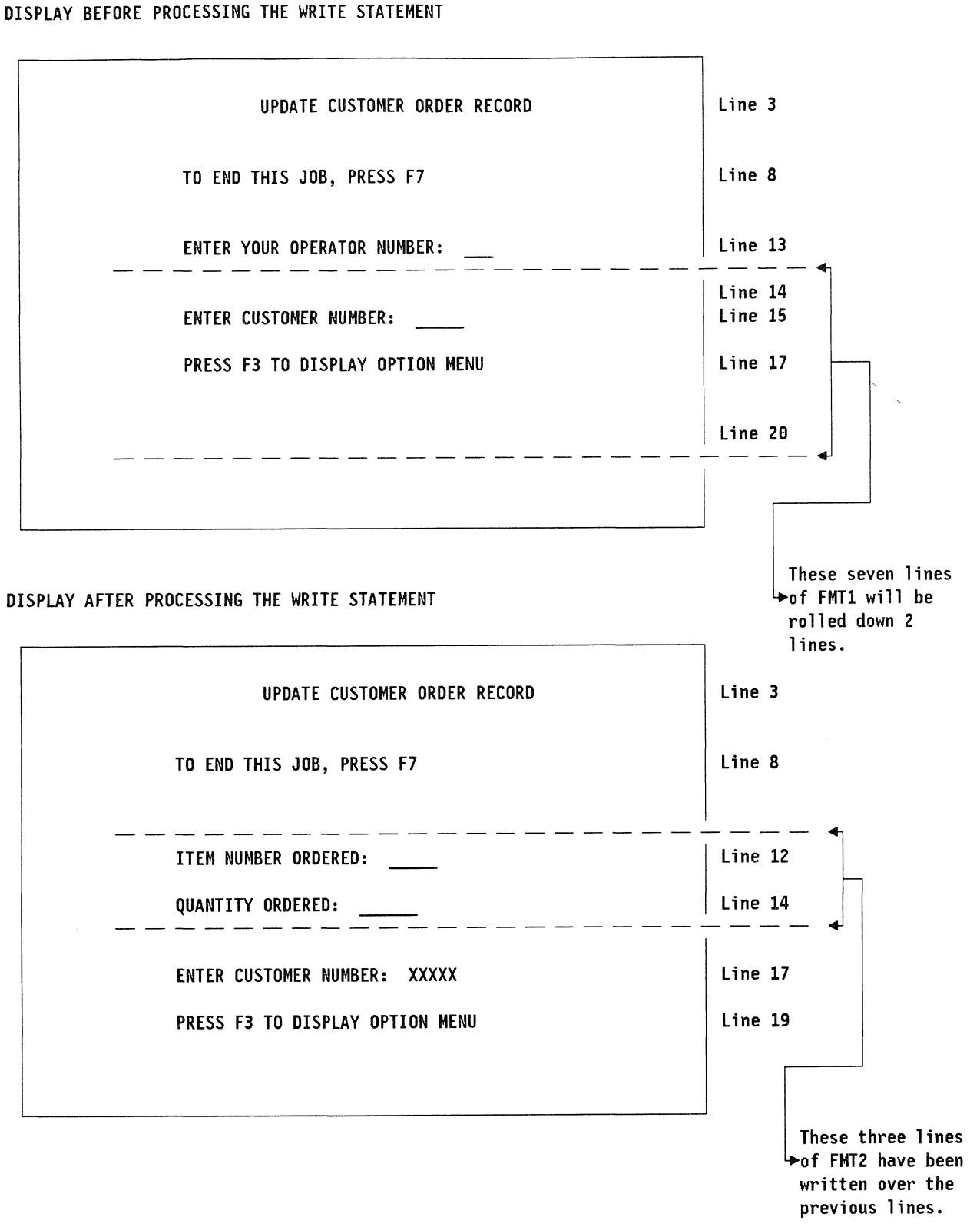
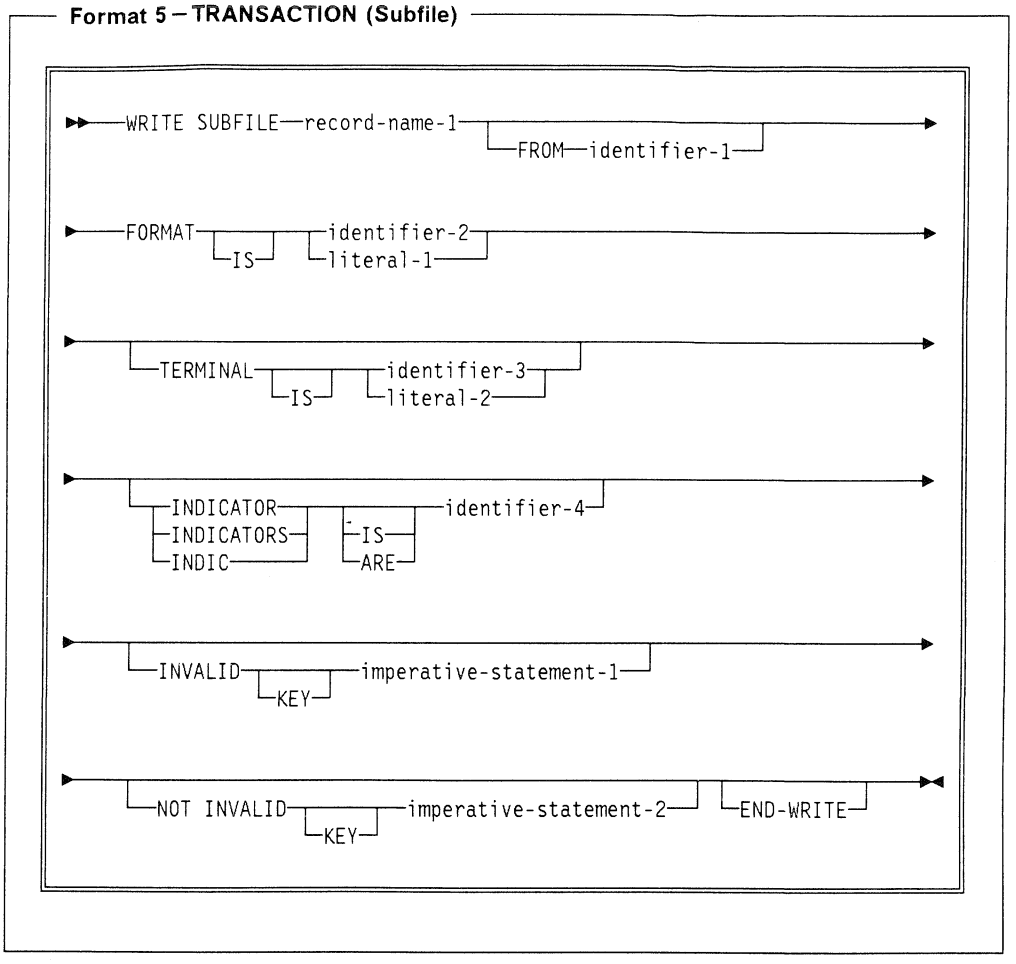


Figure 31. Example of ROLLING Operation

Format 5 — TRANSACTION (Subfile)



Format 5 can only be used for display devices. If the subfile form of the WRITE statement is used for any other type of device, the WRITE operation fails and a file status of 90 is set.

If the format is a subfile record, and SUBFILE is specified, the RELATIVE KEY clause must have been specified on the SELECT clause for the file being written. The record written to the subfile is the record in the subfile identified by the format name that has a relative record number equal to the value of the RELATIVE KEY data item. (See the *Data Management Guide*).

TERMINAL Phrase

See Format 4 above for general considerations concerning the TERMINAL phrase.

The TERMINAL phrase specifies which program device's subfile is to have a record written to it. If the TERMINAL phrase is specified, literal-2 or identifier-3 must refer to a work station associated with the TRANSACTION file. If literal-2 or identifier-3 contains a value of blanks, the TERMINAL phrase is treated as if it was not specified. The work station specified by the TERMINAL phrase must have been acquired, either explicitly or implicitly.

If the TERMINAL phrase is omitted, the subfile used is the subfile associated with the default program device.

INVALID KEY Phrase

The INVALID KEY condition exists if a record is already in the subfile with that record number, or if the relative record number specified is greater than the maximum allowable subfile record number. The INVALID KEY phrase should be specified in the WRITE SUBFILE statement for all files for which an appropriate USE procedure is not specified.

NOT INVALID KEY Phrase

This phrase allows you to specify procedures that will be performed when an invalid key condition does not exist for the statement that is used.

END-WRITE Phrase

This explicit scope terminator serves to delimit the scope of the WRITE statement. END-WRITE permits a conditional WRITE statement to be nested in another conditional statement. END-WRITE may also be used with an imperative WRITE statement.

For more information, see "Delimited Scope Statements" on page 207.

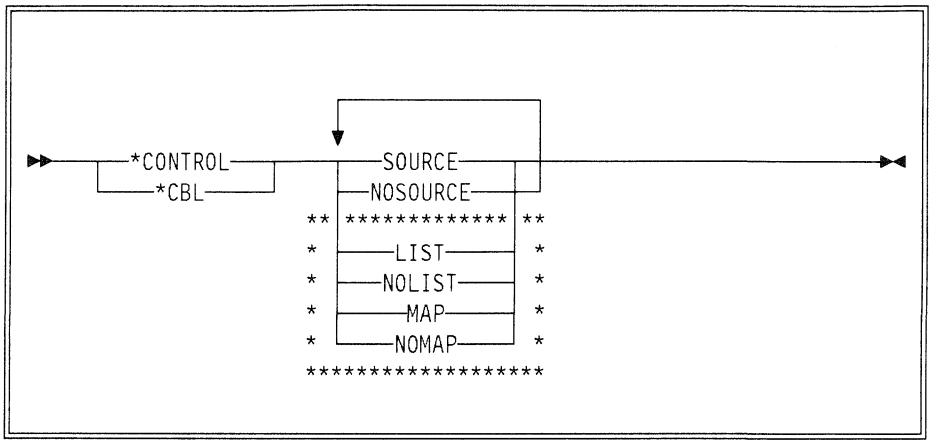
Part 4. Compiler-Directing Statements

IBM Extension

*CONTROL (*CBL) Statement

With the *CONTROL (or *CBL) statement, you can selectively display or suppress the listing of source code throughout the source program.

Format



For a complete discussion of compiler output, see the *COBOL/400 User's Guide*.

The *CONTROL and *CBL statements are synonymous. That is, you can use them interchangeably.

The characters *CONTROL or *CBL can start in any column beginning with column 8, followed by at least one space or comma and one or more option keywords. Separate the option keywords with one or more spaces or commas. This statement must be the only statement on the line, and continuation is not allowed. The statement can end with a period.

The source line containing the *CONTROL (*CBL) statement does not appear in the source listing, but you receive an informational message stating that printing has been suppressed.

The options you request are handled in the following manner:

1. If SOURCE or NOSOURCE appears more than once in a *CONTROL statement, the last occurrence of either option is used.
2. If a *CONTROL NOSOURCE statement is encountered and SOURCE has been requested as a compiler option, printing of the source listing is suppressed from this point on. An informational message is issued stating that printing of the source has been suppressed.

   Afterwards, you can specify *CONTROL SOURCE to resume the printing of the source listing.

   For more information about compiler options, see the *COBOL/400 User's Guide*.

3. If *NOSOURCE is requested as a compiler option, output is *always* inhibited.
4. The *CONTROL statement is in effect only for the source program in which it is written. It does not remain in effect across batch compilation of two or more COBOL source programs.
5. You can use the LIST, NOLIST, MAP, and NOMAP options for compatibility only.

*CONTROL (*CBL) and the COPY Statement

A COPY statement bearing the SUPPRESS phrase overrides any *CONTROL or *CBL options contained in the copied member, but the compiler remembers *CONTROL and *CBL statements that appear in a suppressed COPY member. Once the COPY member has been processed, the last NOSOURCE or SOURCE option in it runs.

If a COPY statement does not bear the SUPPRESS phrase, *CONTROL and *CBL statements within the copied member run immediately.

End of IBM Extension

COPY Statement

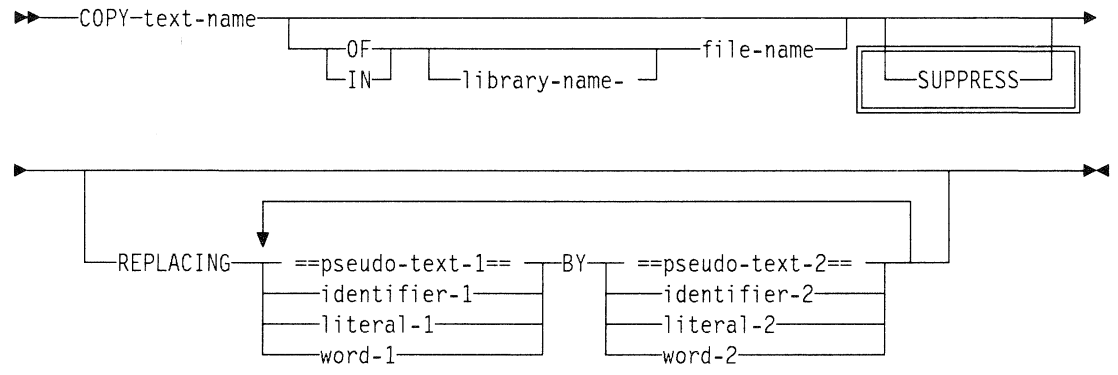
The COPY statement is a library statement that places prewritten text in a COBOL program.

Prewritten source program entries can be included in a source program at compile time. Thus, an installation can use standard file descriptions, record descriptions, or procedures without recoding them. These entries and procedures can then be saved in user-created libraries; they can then be included in the source program by means of the COPY statement.

Compilation of the source program containing COPY statements is logically equivalent to processing all COPY statements before processing the resulting source program.

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 word COPY and ending with the period, inclusive. When the REPLACING phrase is not specified, the library text is copied unchanged.

Format 1



text-name

The text-name is the name of the member to be copied. The text-name must begin with an alphabetic character. The first 10 characters of the text-name are used as the member name; these first 10 characters must, therefore, be unique within one file.

If text-name is not qualified, QLBLSRC is assumed as the file name. If the file name is not qualified by a library name, it is assumed to reside in a library in the library list.

The library name, file name, and text-name must follow the rules for formation of a program-name.

library-name

The first 10 characters of the library-name are used as the identifying name; these first 10 characters must therefore be unique within the system.

Each COPY statement must be preceded by a space and ended with a separator period.

A COPY statement may appear in the source program anywhere a character string or a separator may appear; however, a COPY statement must not be specified within a COPY statement. The resulting copied text must not contain a COPY statement.

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.

When a COPY statement is specified on a debugging line, the copied text is treated as though it appeared on a debugging line, except that comment lines in the text appear as comment lines in the resulting source program.

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.

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 clause is not specified in the SOURCE-COMPUTER paragraph.

Comment lines or blank lines may occur in library text. Comment lines or blank lines appearing in library text are copied into the resultant source program unchanged with the following exception: a 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 (refer to "Replacement and Comparison Rules" on page 459).

The syntactic correctness of the entire COBOL source program cannot be determined until all COPY statements have been completely processed, because the syntactic correctness of the library text cannot be independently determined.

Library text copied from the library is placed into the same area of the resultant program as it is in the library. Library text must conform to the rules for standard COBOL format.

IBM Extension

SUPPRESS Phrase

The SUPPRESS phrase causes a COPY statement to suppress the listing of copied statements. For its duration, this type of COPY statement overrides any *CONTROL or *CBL statement.

If the copied member contains *CONTROL or *CBL statements, the last one runs once the COPY member has been processed.

End of IBM Extension

REPLACING Phrase

In the discussion that follows, each **operand** may consist of one of the following:

- Pseudo-text
- An identifier
- A literal
- A COBOL word (except COPY).

When the REPLACING phrase is specified, the library text is copied, and each properly matched occurrence of operand-1 within the library text is replaced by the associated operand-2.

pseudo-text

A sequence of character-strings and/or separators bounded by, but not including, pseudo-text delimiters (==). Both characters of each pseudo-text delimiter must appear on one line; however, character-strings within pseudo-text can be continued. Because of the replacement rules, the continued line of pseudo-text-2 can begin in Area A.

A character-string for pseudo-text can consist of single-byte characters of any kind. The prefix portion of a data name, however, cannot be replaced using pseudo-text unless the entire data name is used. See Figure 32.

```
Library-member A consists of the following COBOL text:

Area
A         Area
|         B
|         |
|         |
V         V

          10 AA-ID    PIC X(9).

A COPY statement that replaces data description AA-ID with
data description BB-ID, could be written as follows:

Area
A         Area
|         B
|         |
|         |
V         V

                  COPY A REPLACING == AA-ID == BY ==
BB-ID == .
```

Figure 32. Pseudo-Text Continuation Lines

Pseudo-text-1 must not be null, nor may it consist solely of the space character, separator comma, separator semicolon, and/or of comment lines. Beginning and ending blanks are not included in the text comparison process. Embedded blanks are used in the text comparison process to indicate multiple text words.

Pseudo-text-2 may be null; it may consist solely of space characters and/or comment lines. Each text word in pseudo-text-2 that is to be copied into the

program is placed in the same area of the resultant program as the area in which it appears in pseudo-text-2.

identifier
May be defined in any Data Division section.

literal
May be numeric or nonnumeric.

word
May be any single COBOL word (except COPY).

For purposes of matching, each identifier-1, literal-1, or word-1 is treated, respectively, as pseudo-text containing only identifier-1, literal-1, or word-1.

Replacement and Comparison Rules

1. Arithmetic and logical operators are considered text words and may be replaced only through the pseudo-text option.
2. When a figurative constant is operand-1, it will match only if it appears exactly as it is specified. For example, if ALL "AB" is specified in the library text, then "ABAB" is not considered a match; only ALL "AB" is considered a match.
3. Operand-2 is copied in the place of operand-1 unless pseudo-text-2 positioning rules cause the replacement to be inserted in a different area.
4. Any separator comma, semicolon, and/or space preceding the leftmost word in the library text is copied into the source program. Beginning with the leftmost library text word and the first operand-1 specified in the REPLACING option, the entire REPLACING operand that precedes the key word BY is compared to an equivalent number of contiguous library text words.
5. Operand-1 matches the library text if, and only if, the ordered sequence of text words in operand-1 is equal, character for character, to the ordered sequence of library words. For matching purposes, each occurrence of a comma or semicolon separator and each sequence of one or more space separators is considered to be a single space.
6. If no match occurs, the comparison is repeated with each successive operand-1, if specified, until either a match is found or there are no further REPLACING operands.
7. Whenever a match occurs between operand-1 and the library text, the associated operand-2 is copied into the source program.
8. When all operands have been compared and no match is found, the leftmost library text word is copied into the source program.
9. The next successive uncopied library text word is then considered to be the leftmost text word, and the comparison process is repeated, beginning with the first operand-1. The process continues until the rightmost library text word has been compared.
10. Comment lines or blank lines occurring in the library text and in pseudo-text-1 are ignored for purposes of matching; and the sequence of text words in the library text and in pseudo-text-1 is determined by the rules for reference format. 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: a 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.

11. Text words, after replacement, are placed in the source program according to standard COBOL format rules.
12. COPY REPLACING does not affect the EJECT, SKIP1/2/3, or TITLE compiler-directing statements.

Sequences of code (such as file and data descriptions, error and exception routines, etc.) that are common to a number of programs can be saved in a library, and then used in conjunction with the COPY statement. If naming conventions are established for such common code, then the REPLACING phrase need not be specified. If the names will change from one program to another, then the REPLACING phrase can be used to supply meaningful names for this program.

Example 1: In this example, the library text PAYLIB consists of the following Data Division entries:

```
01  A.
  02  B     PIC S99.
  02  C     PIC S9(5)V99.
  02  D     PIC S9999 OCCURS 1 TO 52 TIMES
      DEPENDING ON B OF A.
```

The programmer can use the COPY statement in the Data Division of a program as follows:

```
    COPY PAYLIB.
```

In this program, the library text is copied; the resulting text is treated as if it had been written as follows:

```
01  A.
  02  B     PIC S99.
  02  C     PIC S9(5)V99.
  02  D     PIC S9999 OCCURS 1 TO 52 TIMES
      DEPENDING ON B OF A.
```

Example 2: To change some (or all) of the names within the library text, the programmer can use the REPLACING phrase:

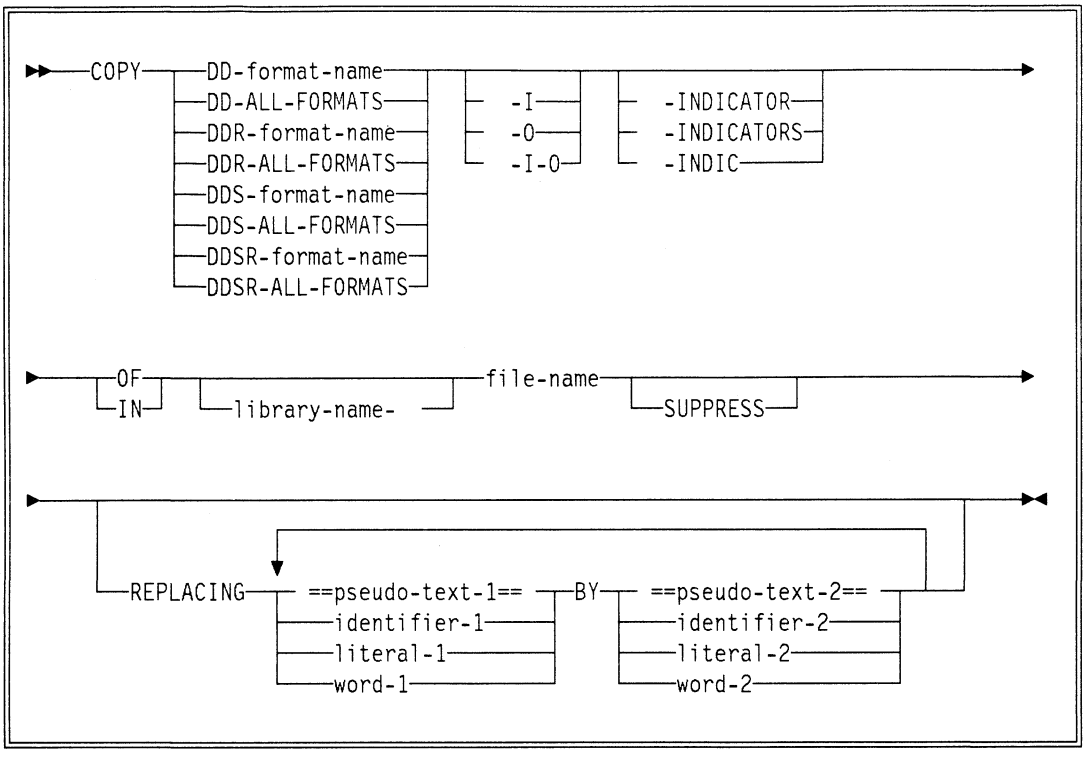
```
  COPY PAYLIB REPLACING  A BY PAYROLL
                         B BY PAY-CODE
                         C BY GROSS-PAY
                         D BY HOURS.
```

In this program, the library text is copied; the resulting text is treated as if it had been written as follows:

```
01  PAYROLL.
  02  PAY-CODE     PIC S99.
  02  GROSS-PAY     PIC S9(5)V99.
  02  HOURS     PIC S9999 OCCURS 1 TO 52 TIMES
      DEPENDING ON PAY-CODE OF PAYROLL.
```

The changes shown are made only for this program. The text, as it appears in the library, remains unchanged.

Format 2—DDS Translate



IBM Extension

Format 2 Considerations

The Format 2 COPY statement (DD, DDR, DDS, or DDSR option) can be used to create COBOL Data Division statements to describe a file that exists on the system. These descriptions are based on the version of the file in existence at compile time. They do not make use of the DDS source statements for the file.

The Format 2 COPY statement can be used only in the Data Division, and it is the user's responsibility to precede the statement with a group level item that has a level-number less than 05.

The DD option is used to reference **alias** (alternate) names. The specification of an alias name in DDS allows a data name of up to 30 characters to be included in the COBOL program.

When the DD option is used, any alias names present replace the corresponding DDS field names. All underscores in the alias names are translated into hyphens before any replacing occurs.

The DDR option does everything that the DD option does. It also replaces the invalid COBOL characters @, #, $, and _ in a field name (or alias name, if applicable) with the corresponding valid COBOL characters A, N, D, and -. As well, it removes underscores from the end of a field name.

The DDS option copies in the internal DDS format field names.

The DDSR option does everything that the DDS option does. It also copies the internal DDS format field names, replacing the invalid COBOL characters @, #, $, and _ with the valid COBOL characters A, N, D, and - accordingly. This option also removes any underscores from the ends of the field names.

The format-name is the name of the DDS record format definition that is to be translated into COBOL data description entries. The format-name must follow the rules for formation of any AS/400 name.

If neither -I nor -O is specified, I-O is assumed.

If a format-name is specified without the indicator attribute, and both -I and -O formats are to be generated, each record format is generated as a redefinition of a 05 elementary item defined as the size of the largest record format that will be generated.

If ALL-FORMATS is specified without the indicator attribute, each record format is generated as a redefinition of a 05 elementary item defined as either:

- the size of the largest record format in the file, if the COPY statement appears in the FILE SECTION.
- the size of the largest record format that will be generated, if the COPY statement appears outside of the FILE SECTION.

When the indicator attribute is specified, **no** redefinition takes place. Instead, each of the formats generates a separate data structure. For details, refer to "INDICATOR Attribute of the Format 2 COPY Statement" on page 466.

If the file is a database file, a single I-O format is generated.

Note: Subfile records with only output or input/output fields, and no field indicators specified, generate I-O formats.

For all other file types the description generated varies as follows:

- If -I is specified, the generated data description entries contain either:
  - The input and input/output fields for a nonsubfile format
  - The input, output, and input/output fields for a subfile format.
- If -O is specified, the generated data description entries contain the output and input/output fields.

The use of the Indicator attribute is discussed under "INDICATOR Attribute of the Format 2 COPY Statement" on page 466.

File-name is the name of an AS/400 system file. The generated DDS entries represent the record format(s) defined in the file. The file must be created before the program is compiled.

Library-name is optional. If it is not specified, the current job library list is used as the default value.

General Notes

- Database files never have indicators.
- When the RECORD KEY clause specifies EXTERNALLY-DESCRIBED-KEY, a format can be copied only once under an FD. For example, if all of the formats of a file are copied under an FD, no other Format 2 COPY statement can be specified for the same file under that FD.
- If a separate storage area is needed in WORKING-STORAGE for each format, an individual COPY statement must be specified for each format.

  For example, if we assume that the file CUSTMASTER contains two formats: CUSTADR and CUSTDETL ; then the following COPY statements could be specified.

```
    SELECT FILE-X
    ASSIGN TO DATABASE-CUSTMASTER.
    .
    .
    .
FD  FILE-X
     LABEL RECORDS ARE STANDARD.
01  FILE-X-RECS.
     COPY DDS-ALL-FORMATS OF
         CUSTMASTER-QGPL.   (See Note 1.)
    .
    .
    .
WORKING-STORAGE SECTION.
01  ADR-REC.
     COPY DDS-CUSTADR OF
         CUSTMASTER.     (See Note 2.)
01  DETAIL-REC.
     COPY DDS-CUSTDETL OF
         CUSTMASTER.     (See Note 2.)
```

Notes:

1. This COPY statement generates only one storage area for all formats.
2. These COPY statements generate separate storage areas.

Data Structures Generated

Format (Record) Level Structures

At the beginning of each format, a table of comments is generated in the source program listing. These comments provide details of the files used during compilation of the program. If there are record keys for the file, comments are also generated to show how the keys are defined in DDS. The record key entries that may appear in the table and the table heading are listed below.

Heading	Possible Entry
NUMBER	key field number
NAME	key field name
RETRIEVAL	ASCENDING, DESCENDING
TYPE	DIGIT, ZONE, ABSVAL, SIGNED, AN (alphanumeric), N (numeric), F (floating point), J (DBCS character), VARLEN (variable length), DDS - L (date), DDS - T (time), DDS - Z (timestamp), DDS - G (fixed-length graphic), G VARLEN (variable-length graphic)
ALTSEQ	NO, YES

If redefinition is required to allow for the generation of multiple formats, a group level name is generated as follows:

```
05 file-name-RECORD
   PIC X(size of largest record).
```

for each format a group level name is assigned as follows:

- INPUT

  05 format-name-I

- OUTPUT

  05 format-name-O

- I-O Format

  05 format-name

Data Field Structures

Field names, PICTURE definitions, and numeric usage clauses are derived directly from the internal DDS format field names (or alias names in the case of the DD option) and data type representations. Field names and PICTURE definitions are constructed as follows:

06 field-name PIC (See Note 1.)

Note: See Figure 33 for the appropriate COBOL definition.

Table 55. Data Field Structures

DDS		COBOL DATA DIVISION n=total field length (DDS pos. 30-34) m=number of decimals (DDS pos. 36 & 37)	
Data Type (pos.35)	Formats	If DDS pos. 36 & 37 are blank	If DDS pos. 36 & 37 are not blank
PHYSICAL, LOGICAL, PRINTER, AND COMMUNICATIONS FILES			
b(Blank)	Default	PIC X(n) [2]	PIC S9(n-m)V9(m)
P	Packed decimal	PIC S9(n) COMP-3	PIC S9(n-m)V9(m) COMP-3
S	Zoned decimal/signed numeric	PIC S9(n)	PIC S9(n-m)V9(m)
B	Binary	PIC S9(n) COMP-4	PIC S9(n-m)V9(m) COMP-4
F	Floating point [1]		
	single precision	PIC 9(5) COMP-4	PIC 9(5) COMP-4
	double precision	PIC 9(10) COMP-4	PIC 9(10) COMP-4
A	Character	PIC X(n) [2]	—
H	Hexadecimal data	PIC X(n)	—
L	Date [3]	PIC X(n)	—
T	Time [3]	PIC X(n)	—
Z	Timestamp [3]	PIC X(n)	—
J	DBCS-Only data	PIC X(n)	—
E	DBCS-Either data	PIC X(n)	—
O	DBCS-Open data	PIC X(n)	—
G	DBCS-Graphic data	PIC X(2n) [2]	—
DISPLAY FILES			
b(Blank)	Default	PIC X(n)	PIC S9(n-m)V9(m)
X	Alphabetic Only	PIC X(n)	—
N	Numeric Shift	PIC X(n)	PIC S9(n-m)V9(m)
Y	Numeric Only	—	PIC S9(n-m)V9(m)
I	Inhibit Keyboard entry	PIC X(n)	PIC S9(n-m)V9(m)
W	Katakana	PIC X(n)	—
A	Alphanumeric Shift	PIC X(n)	—
D	Digits only	PIC X(n)	PIC S9(n)
F	Floating point [1]		
	single precision	PIC 9(5) COMP-4	PIC 9(5) COMP-4
	double precision	PIC 9(10) COMP-4	PIC 9(10) COMP-4
M	Numeric-only character	PIC X(n)	—
S	Signed-numeric shift	—	PIC S9(n-m)V9(m)
E	DBCS-either	PIC X(n)	—
J	DBCS-only	PIC X(n)	—
O	DBCS-open	PIC X(n)	—
G	DBCS-graphic	PIC X(2n)	—

[1] COBOL treats floating point fields as FILLER. See 'Floating Point Fields'.
[2] In DDS, if the field has an attribute of VARLEN, the result is two additional bytes at the beginning of the field.
[3] FILLER items by default. See 'Date, Time, and Timestamp Fields'.

Figure 33. Data Field Structures

Indicator Structures

If indicators are requested, and exist in the format, an additional group name (06 level) is generated at the beginning of the structure, followed by entries (07 level) for the relevant individual indicators.

```
06  format-name-(I or O)-INDIC.
    07  INxx PIC 1 INDIC xx.
```

where xx is the indicator number.

For example:

```
06  SAMPLE1-I-INDIC.
    07  IN01 PIC 1 INDIC 01.
    07  IN04 PIC 1 INDIC 04.
    07  IN05 PIC 1 INDIC 05.
    07  IN07 PIC 1 INDIC 07.
06  FLD1 PIC  ...  .
06  FLD2 PIC  ...  .
```

INDICATOR Attribute of the Format 2 COPY Statement

The INDICATOR attribute specifies whether or not data description entries are generated for indicators.

If the INDICATOR attribute is specified, data description entries are generated for indicators, but not for data fields.

An 05 group level entry is generated as follows:

- If the COPY is for a single structure

  COPY DDS-*format-name*-INDIC
  will generate
  05 *format-name*-I. (or -0 as appropriate).

- If the COPY is for multiple structures

  COPY DDS-ALL-FORMATS-INDIC
  will generate
  05 *file-name*-RECORD.

The data description entries that are generated are determined by which one of the usage attributes (I, O, or I-O) is specified or assumed in the COPY statement.

- If ...I-INDICATOR... is specified, data description entries for input (response) indicators are generated for indicators used in the input record area.
- If ...O-INDICATOR... is specified, data description entries for output (option) indicators are generated for indicators used in the output record area.
- If ...I-O-INDICATOR... is specified or assumed, separate data description entries for both input and output (response and option) indicators are generated for indicators used in the input and output record areas.

The individual indicator descriptions are generated as described under "Indicator Structures."

If the INDICATOR attribute is not specified, whether data description entries are generated for indicators depends on whether the file had the keyword INDARA specified in the DDS at the time it was created.

- If INDARA was not specified, data description entries are generated for both data fields and indicators.
- If INDARA was specified, data description entries are generated for data fields only, not for indicators.

Generation of I-O Formats

When all field descriptions are identical, and you have requested INPUT or OUTPUT fields implicitly or explicitly, only one set of field descriptions is generated. This type of description is annotated with a comment line reading, "I-O FORMAT: format-name". Neither -I nor -O is appended to the record format name.

Note: This always happens for database files because all field descriptions within a database file are identical.

For example:

```
 01 RCUSREC.
    COPY DDS-CUSREC-I OF CUSFILE.
*   I-O FORMAT: CUSREC FROM FILE CUSFILE OF LIBRARY CUSLIB     CUSREC
* THE KEY DEFINITIONS FOR RECORD FORMAT CUSREC
*   NUMBER NAME RETRIEVAL TYPE ALTSEQ
*    0001 ARBAL ASCENDING SIGNED NO
*    0002 AREACD DESCENDING ABSVAL NO
       05   CUSREC.
       06   ARBAL          PIC S9(7)V9(2)      COMP-3          CUSREC
       06   AREACD         PIC S9(3)           COMP-3.         CUSREC
       06   BOSTAZ         PIC X(1).                           CUSREC
       06   CNTCT          PIC X(15).                          CUSREC
       06   CRCHKZ         PIC S9(2).                          CUSREC
       06   CSTAT          PIC X(1).                           CUSREC
       06   CUSTNZ         PIC S9(6).                          CUSREC
       06   DLORD          PIC S9(6).                          CUSREC
       06   DSCPCZ         PIC S9(2)V9(3)      COMP-3.         CUSREC
       06   INDUS          PIC S9(2).                          CUSREC
       06   NAME1          PIC X(25).                          CUSREC
       06   NAME2          PIC X(25).                          CUSREC
       06   NAME3          PIC X(25).                          CUSREC
       06   NAME4          PIC X(25).                          CUSREC
       06   PHONE          PIC S9(7)           COMP-3.         CUSREC
       06   PRICIZ         PIC S9(2).                          CUSREC
       06   SHPINZ         PIC X(25).                          CUSREC
       06   SLSMAZ         PIC X(3).                           CUSREC
       06   TAXCDZ         PIC S9(2).                          CUSREC
       06   TERMSZ         PIC S9(2).                          CUSREC
```

Redefinition of Formats

The user should pay particular attention to the REDEFINES clause that may be generated for the ALL-FORMATS or -I-O phrases. Since all formats are redefined on the same area (generally a buffer area), several field names can describe the same area of storage, and unpredictable results can occur if the entire format area is not reinitialized prior to each output operation.

Data items that are subordinate to the data item specified in a MOVE CORRESPONDING statement do not correspond and are not moved when they contain a REDEFINES clause or are subordinate to a redefining item.

To avoid reinitialization, multiple Format 2 COPY statements (DDS or DD) using -I and -O suffixes can be used to create separate areas of storage in the Working-Storage section for each format or format type (input or output). READ INTO and WRITE FROM statements can be used with these record formats.
For example:

```
FD ORDER-ENTRY-SCREEN . . .
01 ORDER-ENTRY-RECORD . . .
   .
   .
   .
WORKING-STORAGE SECTION.
01 ORDSFL-I-FORMAT.
   COPY DDS-ORDSFL-I OF DOESCR.
01 ORDSFL-O-FORMAT.
   COPY DDS-ORDSFL-O OF DOESCR.
   .
   .
   .
PROCEDURE DIVISION.
   .
   .
   .
READ SUBFILE ORDER-ENTRY-SCREEN NEXT MODIFIED RECORD
   INTO ORDSFL-I-FORMAT FORMAT IS "ORDSFL"
   AT END SET NO-MODIFIED-SUBFILE-RCD TO TRUE.
   .
   .
   .
MOVE CORR ORDSFL-I TO ORDSFL-O.
REWRITE SUBFILE ORDER-ENTRY-RECORD FROM ORDSFL-O-FORMAT
                           FORMAT IS "ORDSFL" . . .
   .
   .
   .
```

Note: The COPY statement can be used in the File Section or in the Working-Storage Section, but the results are not exactly the same. For more information, see "Key Generation Examples" in the *COBOL/400 User's Guide*.

Additional Notes on Field and Format Names

If the generated field name is a COBOL reserved word, the suffix -DDS is added to the field name. If the generated field name originates from a physical file (in other words, the field is an argument of the CONCAT or RENAME keyword), the suffix is also added. For more information, see "Key Generation Examples" in the *COBOL/400 User's Guide*.

The REPLACING phrase cannot be used to change the name of a key field or a format name when EXTERNALLY-DESCRIBED-KEY is used.

Floating Point Fields

COBOL treats floating point fields as FILLER. The fields can contain floating point values set outside of COBOL, and a COMP-4 definition is generated to maintain proper alignment in the record, but the data is *not* in binary format. No attempt must be made to use floating point data for processing in the COBOL program.

Floating point key fields are not allowed. In cases where some formats exist with a floating point key field and other formats do not, you should use one or more Format 2 COPY statements with specific format names, rather than using the ALL-FORMATS option.

Note: If you have not specified your own program collating sequence, you may create a record containing floating point fields in your COBOL program by moving LOW-VALUES to the entire record before moving in the values of the non-floating-point fields. This will give the floating point fields in the record a value of zero. Note that the above method is only recommended if valid floating point fields with a value of zero are desirable for your particular application.

Date, Time, and Timestamp Fields

Date, time, and timestamp fields are brought into your program only if you specify the *DATETIME option of the CRTCBLPGM CVTOPT parameter, or the DATETIME option of the PROCESS statement. Without one of these options, these fields become FILLER items.

Date, time, and timestamp fields are brought in as fixed-length character fields. Your program can perform any valid character operations on them.

The date, time, and timestamp data types each have their own format.

If a field containing date, time, or timestamp information is updated by your program, and the updated information is to be passed back to your database, the format of the field must be exactly the same as it was when the field was retrieved from the database. If you do not use the same format, an error will occur.

Also, if you try to WRITE a record before moving an appropriate value to a date, time, or timestamp field, the WRITE operation will fail with a file status of 90.

For information on valid formats for each data type, see the *DDS Reference*.

Variable-length Fields

You can bring a variable-length field into your program if you specify the *VARCHAR option of the CRTCBLPGM CVTOPT parameter, or the VARCHAR option of the PROCESS statement. A variable-length field that you extract from an externally-described file becomes a fixed-length group item in your program.

See the *COBOL/400 User's Guide* for more detailed information about these fields.

When you perform a WRITE operation before explicitly moving a record to the record area, you will often write blanks, which have a hexadecimal value of 40 (X'40'). For variable-length fields, this means that X'4040' will be used as the current length of the field.

X'4040' translates to a decimal value of 16 448, which would probably exceed the maximum defined length of the variable-length field. This causes the WRITE operation or subsequent CLOSE operation to fail with a file status of 90.

Considerations Regarding Use of REPLACING in Format 2 COPY Statement

The REPLACING phrase can be used to replace any of the generated COBOL source, including the level numbers. (See REPLACING Phrase on page 458 for additional information.) You should, however, note the following exception:

- When RECORD KEY IS EXTERNALLY-DESCRIBED-KEY is specified, the REPLACING phrase cannot change a format-name or the name of a field that is a key.

The following figure describes COPY DDS without the REPLACING option:

```
5738CB1 V2R2M0                    AS/400 COBOL Source
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN  S  COPYNAME   CHG DATE

   1  000100 01  CUSTOMER-RECORD.
      000200*
      000300* COPY DDS  W I T H O U T  REPLACING OPTION
      000400*
   2  000500     COPY DDS-CUSMST OF TESTLIB-CUSMSTP.
     +000001*    I-O FORMAT:CUSMST     FROM FILE CUSMSTP    OF LIBRARY TESTLIB           CUSMST
     +000002*                          ORDER HEADER RECORD                               CUSMST
   3 +000003       05  CUSMST.                                                           CUSMST
   4 +000004           06 CUST              PIC X(5).                                    CUSMST
     +000005*                          CUSTOMER NUMBER                                   CUSMST
   5 +000006           06 NAME              PIC X(25).                                   CUSMST
     +000007*                          CUSTOMER NAME                                     CUSMST
   6 +000008           06 ADDR              PIC X(20).                                   CUSMST
     +000009*                          CUSTOMER ADDRESS                                  CUSMST
   7 +000010           06 CITY              PIC X(20).                                   CUSMST
     +000011*                          CUSTOMER CITY                                     CUSMST
   8 +000012           06 STATE             PIC X(2).                                    CUSMST
     +000013*                          STATE                                             CUSMST
   9 +000014           06 ZIP               PIC S9(5)        COMP-3.                     CUSMST
     +000015*                          ZIP CODE                                          CUSMST
```

Figure 34. COPY DDS Without the REPLACING Option

The following figure describes COPY DDS with the REPLACING option:

```
5738CB1 V2R2M0                    AS/400 COBOL Source
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN  S  COPYNAME   CHG DATE

      000900* COPY DDS  W I T H  REPLACING OPTION
      001000*
   20 001100     COPY DDS-CUSMST OF TESTLIB-CUSMSTP
   21 001200          REPLACING NAME BY ADDR-LINE-1
   22 001300                    ADDR BY ADDR-LINE-2
   23 001400                    CITY BY ADDR-LINE-3.
     +000001*    I-O FORMAT:CUSMST     FROM FILE CUSMSTP    OF LIBRARY TESTLIB          CUSMST
     +000002*                          ORDER HEADER RECORD                              CUSMST
   24 +000003       05  CUSMST.                                                        CUSMST
   25 +000004           06 CUST              PIC X(5).                                 CUSMST
     +000005*                          CUSTOMER NUMBER                                  CUSMST
   26 +000006           06 ADDR-LINE-1             PIC X(25).                          CUSMST
     +000007*                          CUSTOMER NAME                                    CUSMST
   27 +000008           06 ADDR-LINE-2             PIC X(20).                          CUSMST
     +000009*                          CUSTOMER ADDRESS                                 CUSMST
   28 +000010           06 ADDR-LINE-3             PIC X(20).                          CUSMST
     +000011*                          CUSTOMER CITY                                    CUSMST
   29 +000012           06 STATE             PIC X(2).                                 CUSMST
     +000013*                          STATE                                            CUSMST
   30 +000014           06 ZIP               PIC S9(5)        COMP-3.                  CUSMST
     +000015*                          ZIP CODE                                         CUSMST
```

Figure 35. COPY DDS With the REPLACING Option

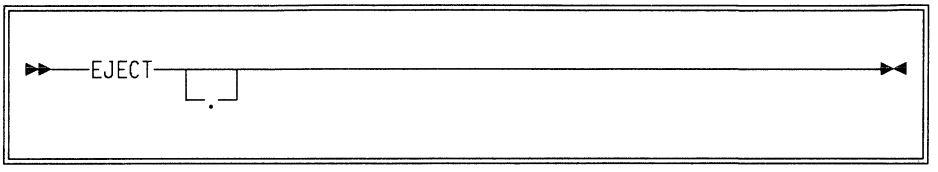
End of IBM Extension

IBM Extension

EJECT Statement

The EJECT statement specifies that the next source statement is to be printed at the top of the next page.

Format



The EJECT statement must be the only statement on the line. It may be written in either Area A or Area B, and may be terminated with a separator period.

The EJECT statement has no effect on the compilation of the source program itself.

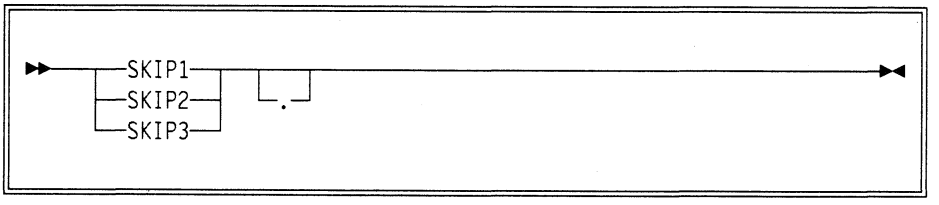
End of IBM Extension

IBM Extension

SKIP1/2/3 Statements

The SKIP1/2/3 statements specify blank lines that the compiler should add when printing the source listing. SKIP statements have no effect on the compilation of the source program itself.

Format



SKIP1
Specifies a single blank line (double spacing).

SKIP2
Specifies two blank lines (triple spacing).

SKIP3
Specifies three blank lines (quadruple spacing).

SKIP1, SKIP2, or SKIP3 causes one occurrence of double, triple, or quadruple spacing.

SKIP1, SKIP2, or SKIP3 may be written anywhere in either Area A or Area B, and may be terminated with a separator period. It must be the only statement on the line.

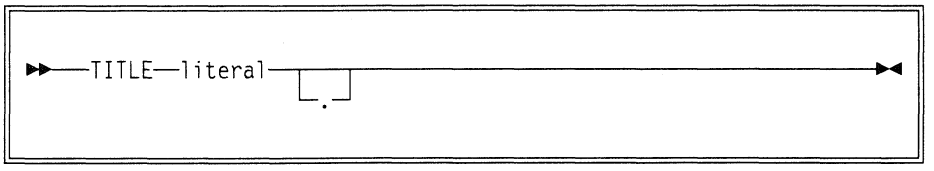
End of IBM Extension

IBM Extension

TITLE Statement

The TITLE statement specifies a title to be printed at the top of each page of the source listing produced during compilation. If no TITLE statement is found, a title containing the identification of the compiler and the current release level is generated. The title is left-justified on the title line.

Format



literal
Must be nonnumeric and may be followed by a separator period. Must not be a figurative constant.

The TITLE statement:

- Forces a new page immediately
- Is not printed on the source listing
- Has no other effect on compilation
- Has no effect on program execution.

A title line is produced for each page in the listing produced by the LIST option. This title line uses the last TITLE statement found in the source statements or the default.

The word TITLE may begin in either Area A or Area B.

The TITLE statement may not be continued on another line.

The TITLE statement may appear anywhere in any of the divisions.

No other statement may appear on the same line as the TITLE statement.

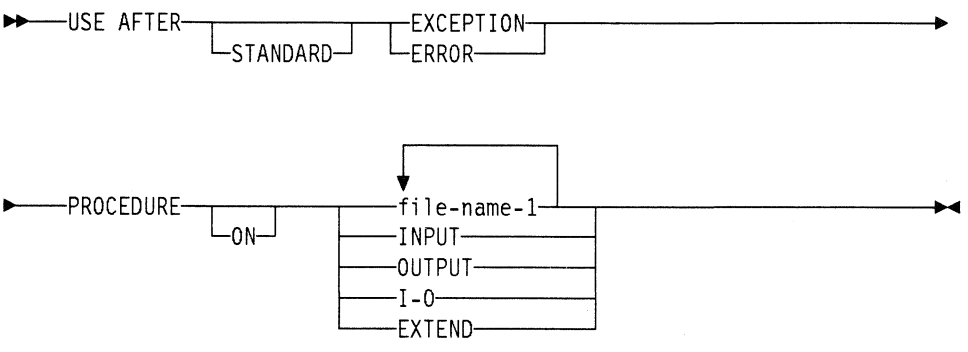
End of IBM Extension

USE Statement

The USE statement specifies procedures for input/output exception or error handling that are to be executed in addition to the system-defined procedures. Although the USE statement is a compiler-directing statement, it can appear only in the Procedure Division, and it can begin only in Area B.

The words EXCEPTION and ERROR are synonymous and may be used interchangeably.

Format



file-name-1
Valid for all files. When this option is specified, the procedure is executed only for the file(s) named. No file-name can refer to a sort or merge file. For any given file, only one EXCEPTION/ERROR procedure may be specified; thus, file-name specification must not cause simultaneous requests for execution of more than one EXCEPTION/ERROR procedure.

IBM Extension

The file-name phrase is also valid for TRANSACTION files.

End of IBM Extension

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.

INPUT
Valid for all files. When this option is specified, the procedure is executed for all files opened in INPUT mode that get an error.

OUTPUT
Valid for all files. When this option is specified, the procedure is executed for all files opened in OUTPUT mode that get an error.

I-O
Valid for all direct-access files. When this option is specified, the procedure is executed for all files opened in I-O mode that get an error.

IBM Extension

The I-O phrase is also valid for TRANSACTION files.

End of IBM Extension

EXTEND
Valid only for files with sequential organization and sequential access mode. When this option is specified, the procedure is executed for all files opened in EXTEND mode that get an error.

The EXCEPTION/ERROR procedure is executed:

- Either after completing the system-defined input/output error routine, or
- Upon recognition of an INVALID KEY or AT END condition when an INVALID KEY or AT END phrase has not been specified in the input/output statement, or
- Upon recognition of an IBM-defined condition that causes status key 1 to be set to 9. (See "Status Key" on page 214.)

The EXCEPTION/ERROR procedures are activated when an input/output error occurs during execution of a ACQUIRE, DROP, READ, WRITE, REWRITE, START, OPEN, CLOSE, or DELETE statement. To determine what conditions are errors, see "Common Processing Facilities" on page 214.

After execution of the EXCEPTION/ERROR Declarative procedure, control is returned to the statement immediately following the input/output statement which caused the error.

Within a declarative procedure, there must be no reference to any nondeclarative procedures. In the nondeclarative portion of the program, there must be no reference to procedure-names that appear in an EXCEPTION/ERROR declarative procedure, except that PERFORM statements may refer to an EXCEPTION/ERROR procedure or to procedures associated with it.

Within an EXCEPTION/ERROR declarative procedure, no statement should be included that would cause execution of a USE procedure that had been previously invoked and had not yet returned control to the invoking routine.

USE Statement Programming Notes

EXCEPTION/ERROR Declarative procedures can be used to check the status key values whenever an input/output error occurs. Additional information about the file causing the error can be obtained by using data from the mnemonic-names OPEN-FEEDBACK and I-O-FEEDBACK.

Care should be used in specifying EXCEPTION/ERROR Declarative procedures for any file. Prior to successful completion of an initial OPEN for any file, the current Declarative has not yet been established by the object program. Therefore, if any other I-O statement is executed for a file that has never been opened, no Declarative can receive control. However, if this file has been previously opened, the last previously established Declarative procedure receives control.

For example, an OPEN OUTPUT statement establishes a Declarative procedure for this file, and the file is then closed without error. During later processing, if a

logic error occurs, control will go to the Declarative procedure established when the file was opened OUTPUT.

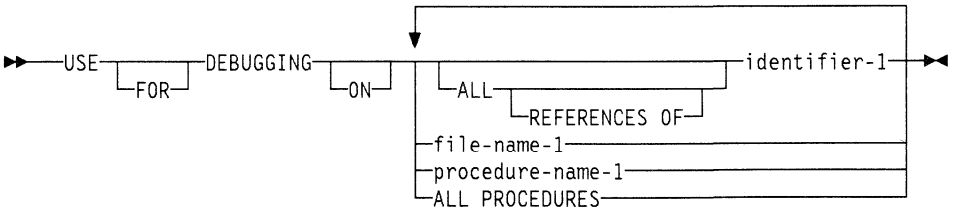
With Standard Error Handling: If there is an applicable file status clause (but not an applicable USE procedure) when an I-O error occurs, the file status is updated, and control returns to the program. In the absence of a file status clause, USE procedure (implicit or explicit), AT END phrase, or INVALID KEY phrase to handle the error, a run-time message is issued, giving you the option to end or return to the program.

Without Standard Error Handling: If there is no applicable USE procedure in the program when an I-O error occurs, processing can continue. Further processing without appropriate intervention can result in severe errors.

USE FOR DEBUGGING

The USE FOR DEBUGGING declarative identifies the items in the source program that are to be monitored by the associated debugging procedure. It establishes a procedure to run when certain errors occur, or when certain items or files change.

Format



Identifier-1 cannot be reference modified.

This statement is compiled only when you are in debugging mode.

The compiler treats all statements that follow this one as comments until the next valid USE AFTER EXCEPTION/ERROR statement or END DECLARATIVES delimiter is reached.

See the *COBOL/400 User's Guide* for further information about this statement.

Part 5. Appendixes

Appendix A. COBOL/400 Compiler Limits

The following table lists the compiler limits supported by the COBOL/400 compiler.

Language Element	COBOL/400 Limit
General	
Number of: Files REPLACING operands in one COPY	 99[2] not defined[1]
Total length of literals	not defined[1]
Total storage available for VALUE clauses	not defined[1]
Number of characters to identify: Library-name Program-name Text-name	 10 10 10
Environment Division	
Number of: SELECT file-names	 99[2]
Maximum number of buffers (areas) specified in the RESERVE clause	not defined[1]
Length of: RECORD KEY in one file	 2 000 bytes
Data Division	
Length of: Working-Storage Section group item Linkage Section group item Elementary item	 3 000 000 bytes[3] 3 000 000 bytes[3] 3 000 000 bytes[3]
Maximum block size	32 766 bytes
Maximum record length	32 766 bytes
Number of: FDs OCCURS levels Levels in data hierarchy SD file-names	 99[2] 7 49 99[2]
Number of: Numeric-edited (data items) character positions Picture character strings Picture replications	 127 30 3 000 000
OCCURS Table size (fixed length) Table size (variable length) Table element size Number of ASC/DESC KEY clauses in one table Total length of ASC/DESC keys in one table INDEXED BY clauses (per table) Pointers in one table	 3 000 000 bytes 32 767 bytes 32 767 bytes not defined[1] not defined[1] not defined[1] not defined[1]

Language Element	COBOL/400 Limit
Procedure Division	
Number of:	
GO TO proc-name DEPENDING ON	255
IF nesting levels	30
CALL parameters	30
SORT-MERGE input files	31
SORT-MERGE keys	30
SEARCH ALL ... WHEN relation conditions	not defined[1]
UNSTRING delimiters	not defined[1]
INSPECT TALLYING identifiers	not defined[1]
INSPECT REPLACING identifiers	not defined[1]
Length of:	
SORT-MERGE keys	256 bytes

Notes to the COBOL/400 compiler limits table:

[1] Limit is determined by system constraints.

[2] Limited to 98 if extended ACCEPT or extended DISPLAY statements are coded.

[3] Limit is 32 767 bytes for the ACCEPT, CANCEL, DISPLAY, EVALUATE, IF, INSPECT, PERFORM, STRING, UNSTRING, and USE statements. For information about how reference modification affects these limits, see page 43.

Appendix B. Summary of IBM Extensions

This appendix contains a brief summary of the COBOL/400 extensions, with references to discussions of the extensions elsewhere in this manual or in the *COBOL/400 User's Guide*.

Character-String Considerations

The apostrophe can be used in place of the quotation mark. See "Characters" in Part 1.

Boolean literals can be used. See "Boolean Literals" in Part 1.

Hexadecimal notation can be used for nonnumeric literals. See "Nonnumeric Literals" in Part 1.

Identification Division

The system uses the first 10 characters of the program-name specified in the PROGRAM-ID paragraph. See "PROGRAM-ID Paragraph" in Part 2.

The abbreviation ID DIVISION may be substituted for the standard division header. See "Identification Division" in Part 2.

Environment Division

The CONSOLE IS CRT clause, the CURSOR IS clause, and the CRT STATUS IS clause may be used in the SPECIAL-NAMES Paragraph for extended ACCEPT or DISPLAY statements.

The CONSOLE IS CRT clause changes the default interpretation of ACCEPT or DISPLAY statements from the ANSI defined format to extended ACCEPT or DISPLAY statements for work station I/O.

The CURSOR IS clause specifies the data item to contain the cursor address used by the ACCEPT Statement.

The CRT STATUS IS clause specifies a data item into which a status value is moved after an ACCEPT statement.

See the "SPECIAL-NAMES Paragraph" in Part 2.

The device that a file will use can be changed at run time with the OVRxxxF CL command. See "FILE-CONTROL Paragraph, ASSIGN Clause" in Part 2.

FORMATFILE must be specified in the ASSIGN clause to use an externally described printer file. See "FILE-CONTROL Paragraph, ASSIGN Clause" in Part 2.

COBOL programs can process database files. The ORGANIZATION of the file indicates the current program usage. See "File Processing Summary" and "FILE-CONTROL Paragraph, ORGANIZATION Clause" in Part 2.

If the DDS keyword DESCEND is used when the field is specified as a key, the records are accessed in the sequence of descending record key values within

the index. The OVRDBF command can set the file position indicator when the file is opened. See "FILE-CONTROL Paragraph, ACCESS MODE Clause" in Part 2.

The RECORD KEY data item, data-name-2, can be numeric when the file is assigned to a DATABASE device type. EXTERNALLY-DESCRIBED-KEY can be specified in the RECORD KEY clause. See "FILE-CONTROL Paragraph, RECORD KEY Clause" in Part 2.

The DUPLICATES phrase can only be specified for files assigned to DATABASE. See "FILE-CONTROL Paragraph, RECORD KEY Clause" in Part 2. Additional information is given in discussions of the READ, REWRITE, DELETE, and WRITE statements in Part 3.

The keywords specified for the data item in DDS can modify record sequence. See "FILE-CONTROL Paragraph, RECORD KEY Clause" in Part 2.

The COMMITMENT CONTROL clause can be specified to enable the synchronizing or canceling of database changes, and to provide additional record locking for records being changed. See "COMMITMENT CONTROL Clause" in Part 2 and the section on Commitment Control in the *COBOL/400 User's Guide*.

Data Division

A Boolean data type has been provided as a means of modifying and passing the values of the indicators associated with the display screen formats. See "Data Description Entry — Boolean Data" in Part 2.

The Boolean symbol 1 is allowed once in the PICTURE clause. See "PICTURE Clause" in Part 2.

Elementary items or group items immediately subordinate to one group item can have unequal level-numbers. See "Data Relationships" in Part 2.

The OVRTAPF command can change the LABEL RECORDS clause at run time. See "File Description Entry, LABEL RECORDS Clause" in Part 2.

If the CODE-SET clause is omitted, the CODE parameter of the CRTDKTF or the CRTTAPF command is used. The OVRDKTF or the OVRTAPF command can change the CODE-SET clause at run time. See "File Description Entry, CODE-SET Clause" in Part 2.

COMPUTATIONAL-3 (internal decimal) and COMPUTATIONAL-4 (binary) can be specified for the USAGE clause of numeric items. See "USAGE Clause" in Part 2.

The key specified for an OCCURS clause can have USAGE of COMPUTATIONAL-3 or COMPUTATIONAL-4. See "USAGE Clause" in Part 2.

The JUSTIFIED clause can be specified for alphanumeric edited items. See "JUSTIFIED Clause" in Part 2.

The LIKE clause can be used to copy characteristics from a previously defined data item. See "LIKE Clause" in Part 2.

Pointer data items allow easier migration of COBOL code from other SAA platforms, and allow access to Application Programming Interfaces (APIs) and languages that require pointers. See "POINTER Phrase" in Part 2.

Procedure Division

Work station I/O may be performed through the extended ACCEPT or DISPLAY statements. See "ACCEPT Statement - Format 7" and "DISPLAY Statement - Format 3" in Part 3.

The mnemonic-names OPEN-FEEDBACK and I-O-FEEDBACK are used for file information and are accessed through an ACCEPT statement format. See "ACCEPT Statement" in Part 3.

The GOBACK statement can be used to specify the logical end of a called program. See "GOBACK Statement" in Part 3.

The FORMAT phrase is valid for DELETE, READ, REWRITE, START, and WRITE statements. See "DELETE Statement", "READ Statement", "REWRITE Statement", "START Statement", and "WRITE Statement" in Part 3.

The special register DB-FORMAT-NAME contains information about the processing of file input/output statements. See "DB-FORMAT-NAME Special Register" in Part 3.

The identifier in a Format 2 ACCEPT statement can be an internal decimal item. See "ACCEPT Statement" in Part 3.

The ACCEPT statement can be used to transfer data from a job's local data area to a specified data item. See "ACCEPT Statement" in Part 3 and the section on the Local Data Area in the *COBOL/400 User's Guide*.

The ACCEPT statement can be used to transfer data from a PIP (Program Initialization Parameters) data area into an identifier item. See "ACCEPT Statement" in Part 3 and the section on the PIP Data Area in the *COBOL/400 User's Guide*.

The system always rewinds and unloads the tape when REEL/UNIT is specified in the CLOSE statement. See "CLOSE Statement" in Part 3.

The INHWRT parameter of the OVRDBF command can inhibit the DELETE, READ, and WRITE statements. See "DELETE Statement", "READ Statement", and "WRITE Statement" in Part 3.

For a file with duplicate primary keys allowed, a successfully completed READ statement must immediately precede a DELETE or REWRITE statement to ensure proper deletion. See "DELETE Statement" and "REWRITE Statement" in Part 3.

For the DISPLAY statement, signed noninteger numeric literals are allowed. See "DISPLAY Statement" in Part 3. In a Format 1 DISPLAY statement, environment-name may be specified in place of mnemonic-name. See "DISPLAY Statement" in Part 3.

The DISPLAY statement can be used to transfer data to a job's local data area. See "DISPLAY Statement" in Part 3 and the section on the Local Data Area in the *COBOL/400 User's Guide*.

A logical file opened for OUTPUT does not remove all records in the physical file on which it is based; the records are added to the end of the file. The OVRDBF command can specify the first record to be made available to the program at run time. See "OPEN Statement" in Part 3.

FIRST, PRIOR, and LAST can be specified in the READ statement for indexed files with dynamic access. See "READ Statement" in Part 3.

NO LOCK can be specified in the READ statement for files opened in I-O (update) mode. See "READ Statement" in Part 3.

EXTERNALLY-DESCRIBED-KEY can be specified through the KEY phrase of the START statement. A comparison can be affected by the type of key fields in the record area defined for the file. See "START Statement" in Part 3.

The composite of all operands in an arithmetic statement can have a maximum of 30 digits. See "Arithmetic Statement Operands" in Part 2.

In the CORRESPONDING phrase of an ADD or SUBTRACT statement, the identifiers d1 and d2 can be subordinate to a FILLER item. See "ADD Statement" and "SUBTRACT Statement" in Part 3.

Two active PERFORM statements can have a common exit point. See "PERFORM Statement" in Part 3.

Input files do not need to be sequenced before a sort or merge operation. See "MERGE Statement" and "SORT Statement" in Part 3.

The COMMIT statement can be used to synchronize changes to records in database files under commitment control, while preventing other jobs from accessing or modifying those records until the COMMIT is complete. See "COMMIT Statement" in Part 3.

The ROLLBACK statement can be used to cancel database changes from files under commitment control when the changes should not remain permanent. See "ROLLBACK Statement" in Part 3.

The CALL GDDM statement lets you access the OS/400 graphics routines, Graphical Data Display Manager (GDDM), and Presentation Graphics Routines (PGR). See "OS/400 Graphics Support" in Part 3.

The WHEN-COMPILED special register (a 16-character alphanumeric data-item) makes available to the object program the date-and-time-compiled constant in the object module at the start of compilation; it is valid only as the sending item in a MOVE statement. See "MOVE Statement" in Part 3.

The ADDRESS OF phrase can be used to refer to the calculated address a data item. See "ADDRESS OF" in Part 2.

The ADDRESS OF special register returns a pointer containing the address of an item in the Linkage Section of a program. See "ADDRESS OF Special Register" in Part 2.

The LENGTH OF special register eliminates the need to refer to the exact length of a data item. See "LENGTH OF Special Register" in Part 1.

The NULL figurative constant returns a pointer whose address is not valid. See "VALUE Clause" in Part 2.

COPY Statement—All Divisions

The file-name is optional for the Format 1 COPY Statement. The default file-name is QLBLSRC. See "Qualification Rules" in Part 4.

The COPY statement (DD, DDR, DDS, or DDSR option) is used to specify record description entries so that field descriptions for a record format are exactly as defined in DDS. See "Data Division" in Part 2 and "Compiler Directing Statements" in Part 4.

Transaction Files

The data organization for work stations, programs, devices on a remote system, or any combination of the above, is TRANSACTION. Transaction files have special formats for the file-control entry, file description entry, and the input/output statements. Transaction file considerations are covered in the *COBOL/400 User's Guide*.

Considerations for the TRANSACTION file-control entry include those for:

- The ASSIGN clause
- The ORGANIZATION clause
- The ACCESS MODE clause
- The FILE STATUS clause
- The CONTROL-AREA clause.

Considerations for the TRANSACTION file description entry are the same as those for other file description entries.

The ACCEPT statement provides a way of accessing information about a program device when function-name is associated with a mnemonic-name of ATTRIBUTE-DATA in the SPECIAL-NAMES paragraph. See "ACCEPT Statement, Format 5 Considerations" in Part 3.

TRANSACTION file considerations for OPEN, CLOSE, READ, WRITE, REWRITE, and USE statements are given under the discussions for the respective statements. A section on Transaction Files is provided in the *COBOL/400 User's Guide*.

The ACQUIRE statement can be used to acquire a program device for a transaction file. See "ACQUIRE Statement" in Part 3.

The DROP statement can be used to release a program device acquired by a transaction file. See "DROP Statement" in Part 3.

Compiler-Directing Statements

The *CONTROL or *CBL statement can be used to selectively display or suppress the listing of source code throughout the source program.

The SUPPRESS phrase can be used in the COPY statement to suppress the listing of source statements. See "COPY Statement" in Part 4.

The Format 2 COPY statement can be used to create COBOL Data Division statements that describe an existing file in the system. See "COPY Statement" in Part 4.

The EJECT, SKIP1, SKIP2, SKIP3, and, TITLE statements have been provided to allow formatting control over program listings. See "EJECT Statement," "SKIP1/2/3 Statements," and "TITLE Statement" in Part 4.

Comment-entry may contain the EJECT, SKIP1, SKIP2, SKIP3, or TITLE statement anywhere on the line. See "Optional Paragraphs" in Part 1.

Appendix C. Intermediate Result Fields

This appendix discusses the conceptual compiler algorithms for determining the number of integer and decimal places reserved for intermediate results. The following abbreviations are used:

i	Number of integer places carried for an intermediate result.
d	Number of decimal places carried for an intermediate result.
dmax	In a particular statement the larger of either: • The number of decimal places needed for the final result field(s) • The maximum number of decimal places defined for any operand except exponents and divisors.
op1	First operand in a generated arithmetic statement.
op2	Second operand in a generated arithmetic statement.
i1,i2	Number of integer places in op1 and op2, respectively.
d1,d2	Number of decimal places defined for op1 or op2, respectively.
ir	Intermediate result field obtained from the processing of a generated arithmetic statement or operation. Intermediate results are represented by ir1, ir2, and so on. Successive intermediate results may share the same memory location.

When an arithmetic statement contains only a single pair of operands, no intermediate results are generated. Intermediate results are possible in the following cases:

- In an ADD or SUBTRACT statement containing multiple operands immediately following the verb
- In a COMPUTE statement specifying a series of arithmetic operations
- In arithmetic expressions contained in an IF or PERFORM statement
- In the GIVING option with multiple result fields for the ADD, SUBTRACT, MULTIPLY, DIVIDE, or COMPUTE statements.

In such cases, the compiler treats the statement as a succession of operations. For example, the following statement:

```
COMPUTE Y = A + B * C - D / E + F ** G
```

is replaced by

```
F ** G                                  yielding ir1
MULTIPLY B          BY C                yielding ir2
DIVIDE E            INTO D              yielding ir3
ADD A               TO ir2              yielding ir4
SUBTRACT ir3        FROM ir4            yielding ir5
ADD ir5             TO ir1              yielding Y
```

Compiler Calculation of Intermediate Results

The number of integer places in an ir is calculated as follows:

The compiler first determines the maximum value that the ir can contain by assigning a numerical value to each of the operands used to generate the ir, and determining the value that would result from the operation.

- If an operand in this statement is a data-name, the value used for the data-name is equal to the numerical value of the PICTURE for the data-name (that is, PICTURE 9V99 has the value 9.99).
- If an operand is a literal, the literal is treated as though it had a PICTURE, and the numerical value of the PICTURE is used (that is, the literal +127.3 has an implied PICTURE S999V9).
- If an operand is an intermediate result, the PICTURE determined for the intermediate result in a previous operation is used. The numerical value of that PICTURE is used.
- If the operation is division:
  - If op2 is a data-name, the value used for op2 is the minimum nonzero value of the digit in the PICTURE for the data-name (that is, PICTURE 9V99 has the value 0.01).
  - If op2 is an intermediate result, the intermediate result is treated as though it had a PICTURE, and the minimum nonzero value of the digits in this PICTURE is used.

Once the maximum value of the ir has been determined by the above procedures, i is set equal to the number of integers in the maximum value.

The number of decimal places contained in an ir is calculated as:

Operation	Integer Places	Decimal Places
+ or −	(i1 or i2) + 1, whichever is greater	d1 or d2, whichever is greater
*	i1 + i2	d1 + d2
/	i1 + d2	d1 - d2 or dmax, whichever is greater
**	when i2 = 0 max(min(i1,18),1) if op2 is nonintegral[1] max(min(i1 * i1,18),1) if op2 is an integral literal[1] when i2 ≠ 0 max(min(i1 * (9 * i2),18),1) if op2 is nonintegral[1] max(min(i1 * i1 * (9 * i2),18),1) if op2 is an integral literal[1]	dmax if op2 is nonintegral or a data-name; d1 * op2 if op2 is an integral literal

[1] These results are subject to subsequent processing.

Note: The user must define the operands of any arithmetic statement with enough decimal places to give the desired accuracy in the final result.

Table 56 indicates the action of the compiler when handling intermediate results.

Table 56. Compiler Action on Intermediate Results

Value of i + d	Value of d	Value of i + dmax	Action Taken
<30 = 30	Any value	Any value	*i* integer and *d* decimal places are carried for ir
>30	<dmax =dmax	Any value	30 - *d* integer and *d* decimal places are carried for ir
	>dmax	<30 = 30	*i* integer and 30 - *i* decimal places are carried for ir
		>30	30 - dmax integer and dmax decimal places are carried for ir

Notes

- If the value of i + d exceeds 31, system message MCH1202 can result, even if the statement includes the SIZE ERROR phrase.
- If the value of i + d is an even number less than 30, the compiler converts it to an odd number by adding 1.

Appendix D. EBCDIC and ASCII Collating Sequences

The ascending collating sequences for both the EBCDIC (Extended Binary Coded Decimal Interchange Code) and ASCII (American National Standard Code for Information Interchange) character sets are shown in this appendix. In addition to the symbol and meaning for each character, the ordinal number (beginning with 1), decimal representation, and hexadecimal representation are given.

EBCDIC Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
65	b	Space	64	40
. . .				
75	¢	Cent sign	74	4A
76	.	Period, decimal point	75	4B
77	<	Less than sign	76	4C
78	(	Left parenthesis	77	4D
79	+	Plus sign	78	4E
80	|	Vertical bar, logical OR	79	4F
81	&	Ampersand	80	50
. . .				
91	!	Exclamation point	90	5A
92	$	Dollar sign	91	5B
93	*	Asterisk	92	5C
94	)	Right parenthesis	93	5D
95	;	Semicolon	94	5E
96	¬	Logical NOT	95	5F
97	-	Minus, hyphen	96	60
98	/	Slash	97	61
. . .				
108	,	Comma	107	6B
109	%	Percent sign	108	6C
110	_	Underscore	109	6D
111	>	Greater than sign	110	6E
112	?	Question mark	111	6F
. . .				

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
123	:	Colon	122	7A
124	#	Number sign, pound sign	123	7B
125	@	At sign, circa sign	124	7C
126	'	Apostrophe, prime sign	125	7D
127	=	Equal sign	126	7E
128	"	Quotation marks	127	7F
. . .				
130	a		129	81
131	b		130	82
132	c		131	83
133	d		132	84
134	e		133	85
135	f		134	86
136	g		135	87
137	h		136	88
138	i		137	89
. . .				
146	j		145	91
147	k		146	92
148	l		147	93
149	m		148	94
150	n		149	95
151	o		150	96
152	p		151	97
153	q		152	98
154	r		153	99
. . .				
163	s		162	A2
164	t		163	A3
165	u		164	A4
166	v		165	A5
167	w		166	A6
168	x		167	A7
169	y		168	A8
170	z		169	A9

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
. . .				
194	A		193	C1
195	B		194	C2
196	C		195	C3
197	D		196	C4
198	E		197	C5
199	F		198	C6
200	G		199	C7
201	H		200	C8
202	I		201	C9
. . .				
210	J		209	D1
211	K		210	D2
212	L		211	D3
213	M		212	D4
214	N		213	D5
215	O		214	D6
216	P		215	D7
217	Q		216	D8
218	R		217	D9
. . .				
227	S		226	E2
228	T		227	E3
229	U		228	E4
230	V		229	E5
231	W		230	E6
232	X		231	E7
233	Y		232	E8
234	Z		233	E9
. . .				
241	0		240	F0
242	1		241	F1
243	2		242	F2
244	3		243	F3

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
245	4		244	F4
246	5		245	F5
247	6		246	F6
248	7		247	F7
249	8		248	F8
250	9		249	F9

ASCII Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
1		Null	0	0
. . .				
33	b	Space	32	20
34	!	Exclamation point	33	21
35	"	Quotation mark	34	22
36	#	Number sign	35	23
37	$	Dollar sign	36	24
38	%	Percent sign	37	25
39	&	Ampersand	38	26
40	'	Apostrophe, prime sign	39	27
41	(	Left parenthesis	40	28
42	)	Right parenthesis	41	29
43	*	Asterisk	42	2A
44	+	Plus sign	43	2B
45	,	Comma	44	2C
46	-	Hyphen, minus	45	2D
47	.	Period, decimal point	46	2E
48	/	Slash	47	2F
49	0		48	30
50	1		49	31
51	2		50	32
52	3		51	33
53	4		52	34
54	5		53	35
55	6		54	36
56	7		55	37
57	8		56	38

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
58	9		57	39
59	:	Colon	58	3A
60	;	Semicolon	59	3B
61	<	Less than sign	60	3C
62	=	Equal sign	61	3D
63	>	Greater than sign	62	3E
64	?	Question mark	63	3F
65	@	At sign, circa sign	64	40
66	A		65	41
67	B		66	42
68	C		67	43
69	D		68	44
70	E		69	45
71	F		70	46
72	G		71	47
73	H		72	48
74	I		73	49
75	J		74	4A
76	K		75	4B
77	L		76	4C
78	M		77	4D
79	N		78	4E
80	O		79	4F
81	P		80	50
82	Q		81	51
83	R		82	52
84	S		83	53
85	T		84	54
86	U		85	55
87	V		86	56
88	W		87	57
89	X		88	58
90	Y		89	59
91	Z		90	5A
92	[	Left bracket	91	5B
93	\	Reverse slash	92	5C
94	]	Right bracket	93	5D
95	^	Circumflex accent, caret	94	5E
96	_	Underscore	95	5F
97	`	Grave accent, right prime	96	60

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
98	a		97	61
99	b		98	62
100	c		99	63
101	d		100	64
102	e		101	65
103	f		102	66
104	g		103	67
105	h		104	68
106	i		105	69
107	j		106	6A
108	k		107	6B
109	l		108	6C
110	m		109	6D
111	n		110	6E
112	o		111	6F
113	p		112	70
114	q		113	71
115	r		114	72
116	s		115	73
117	t		116	74
118	u		117	75
119	v		118	76
120	w		119	77
121	x		120	78
122	y		121	79
123	z		122	7A
124	{	Left brace	123	7B
125	¦	Split vertical bar	124	7C
126	}	Right brace	125	7D
127	~	Tilde	126	7E

Appendix E. COBOL/400 Reserved Word List

The following key identifies the reserved words in Version 2 Release 2 of the COBOL/400 language:

Blank A COBOL/400 reserved word from the 1985 ANSI Standard.

[1] A COBOL/400 reserved word that is an IBM extension to the 1985 ANSI Standard.

[2] A COBOL reserved word from the 1985 (revised 1989) ANSI Standard that is not used by the COBOL/400 compiler. These words should not be used if compatibility is important to an installation. If used, a diagnostic message will be issued.

[3] A CODASYL COBOL reserved word that is not in the 1985 ANSI Standard and is not supported by the COBOL/400 compiler. If used, a diagnostic message will be issued.

[4] An SAA COBOL reserved word that is not in the 1985 ANSI Standard and is not supported by the COBOL/400 compiler. If used, a diagnostic message will be issued.

[5] A COBOL reserved word that is supported by the COBOL/400 compiler when *EXTACCDSP is specified in the CRTCBLPGM CL command, or when EXTACCDSP is used in the PROCESS statement.

[6] A COBOL reserved word from the 1985 ANSI standard that is not used by the COBOL/400 compiler unless *EXTACCDSP or EXTACCDSP is specified. If the word is used in the absence of these options, a diagnostic message will be issued.

Reserved Word
ACCEPT
ACCESS
ACQUIRE [1]
ADD
ADDRESS [1]
ADVANCING
AFTER
ALL
ALPHABET
ALPHABETIC
ALPHABETIC-LOWER
ALPHABETIC-UPPER
ALPHANUMERIC
ALPHANUMERIC-EDITED
ALSO
ALTER
ALTERNATE
AND
ANY [2]
ARE
AREA
AREAS
ARITHMETIC [3]
ASCENDING
ASSIGN
AT
AUTHOR
AUTO [5]
AUTO-SKIP [5]
BACKGROUND-COLOR [5]
BACKGROUND-COLOUR [5]

Reserved Word
B-AND [3]
BEEP [5]
BEFORE
BELL [5]
B-EXOR [3]
BINARY
BIT [3]
BITS [3]
BLANK
B-LESS [3]
BLINK [5]
BLOCK
B-NOT [3]
BOOLEAN [3]
B-OR [3]
BOTTOM
BY
CALL
CANCEL
CD [2]
CF [2]
CH [2]
CHARACTER
CHARACTERS
CLASS
CLOCK-UNITS [2]
CLOSE
COBOL [2]
CODE [2]
CODE-SET
COL [5]

Reserved Word
COLLATING
COLUMN [5] [6]
COMMA
COMMIT [1]
COMMITMENT [1]
COMMON [2]
COMMUNICATION [2]
COMP
COMP-3
COMP-4
COMPUTATIONAL
COMPUTATIONAL-3
COMPUTATIONAL-4
COMPUTE
CONFIGURATION
CONNECT [3]
CONTAINED [3]
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROL-AREA [1]
CONTROLS
CONVERSION [3]
CONVERTING
COPY
CORR
CORRESPONDING
COUNT
CRT [5]
CRT-UNDER [5]

Reserved Word
CURRENCY
CURRENT [3]
CURSOR [5]
DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DAY-OF-WEEK [2]
DB [3]
DB-ACCESS-CONTROL-KEY [3]
DB-DATA-NAME [3]
DB-EXCEPTION [3]
DB-FORMAT-NAME [1]
DB-RECORD-NAME [3]
DB-SET-NAME [3]
DB-STATUS [3]
DBCS [4]
DE [2]
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DEFAULT [3]
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION [2]
DETAIL [2]
DISABLE [2]
DISCONNECT [3]
DISPLAY
DISPLAY-1 [3]
DISPLAY-n [3]
DIVIDE
DIVISION
DOWN
DROP [1]
DUPLICATE [3]
DUPLICATES
DYNAMIC
EGI [2]
EJECT [1]
ELSE
EMI [2]
EMPTY-CHECK [5]
ENABLE [2]
END
END-ACCEPT [1]
END-ADD
END-CALL
END-COMPUTE
END-DELETE
END-DIVIDE
END-EVALUATE
END-IF
END-MULTIPLY
END-OF-PAGE
END-PERFORM
END-READ
END-RECEIVE [2]

Reserved Word
END-RETURN
END-REWRITE
END-SEARCH
END-START
END-STRING
END-SUBTRACT
END-UNSTRING
END-WRITE
ENTER
ENVIRONMENT
EOP
EQUAL
EQUALS [3]
ERASE [3]
ERROR
ESI [2]
EVALUATE
EVERY
EXCEEDS [3]
EXCEPTION
EXCLUSIVE [3]
EXIT
EXOR [3]
EXTEND
EXTERNAL [2]
EXTERNALLY-DESCRIBED-KEY [1]
FALSE [2]
FD
FETCH [3]
FILE
FILE-CONTROL
FILES [3]
FILLER
FINAL [2]
FIND [3]
FINISH [3]
FIRST
FOOTING
FOR
FOREGROUND-COLOR [5]
FOREGROUND-COLOUR [5]
FORMAT [1]
FREE [3]
FROM
FULL [5]
FUNCTION [2]
GENERATE
GET [3]
GIVING
GLOBAL [2]
GO
GOBACK [1]
GREATER
GROUP [2]
HEADING [2]
HIGHLIGHT [5]
HIGH-VALUE
HIGH-VALUES
I-O
I-O-CONTROL
ID [1]
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDEX-N [3]
INDIC [1]
INDICATE

Reserved Word
INDICATOR [1]
INDICATORS [1]
INITIAL
INITIALIZE
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS
JUST
JUSTIFIED
KEEP [3]
KEY
LABEL
LAST
LD [3]
LEADING
LEFT
LEFT-JUSTIFY [5]
LENGTH
LENGTH-CHECK [5]
LESS
LIKE [1]
LIMIT [2]
LIMITS [2]
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER [2]
LINES
LINKAGE
LOCALLY [3]
LOCK
LOW-VALUE
LOW-VALUES
MEMBER [3]
MEMORY
MERGE
MESSAGE [2]
MODE
MODIFIED [1]
MODIFY [3]
MODULES
MOVE
MULTIPLE
MULTIPLY
NATIVE
NEGATIVE
NEXT
NO
NO-ECHO [5]
NONE [3]
NOT
NULL [1]
NULLS [1]
NUMBER [5 6]
NUMERIC
NUMERIC-EDITED
OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
ONLY [3]
OPEN

Reserved Word
OPTIONAL
OR
ORDER
ORGANIZATION
OTHER [2]
OUTPUT
OVERFLOW
OWNER [3]
PACKED-DECIMAL
PADDING [2]
PAGE
PAGE-COUNTER [2]
PERFORM
PF [2]
PH [2]
PIC
PICTURE
PLUS [2]
POINTER
POSITION
POSITIVE
PRESENT [3]
PRINTING
PRIOR [1]
PROCEDURE
PROCEDURES
PROCEED
PROCESS [1]
PROGRAM
PROGRAM-ID
PROMPT [5]
PROTECTED [3]
PURGE [2]
QUEUE [2]
QUOTE
QUOTES
RANDOM
RD [2]
READ
READY [3]
REALM [3]
RECEIVE [2]
RECONNECT [3]
RECORD
RECORD-NAME [3]
RECORDS
REDEFINES
REEL
REFERENCE
REFERENCE-MONITOR [3]
REFERENCES
RELATION [3]
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPEATED [3]
REPLACE [2]
REPLACING
REPORT [2]
REPORTING [2]
REPORTS [2]
REQUIRED [5]
RERUN
RESERVE
RESET [2]
RETAINING [3]
RETRIEVAL [3]

Reserved Word
RETURN
RETURN-CODE [4]
REVERSED
REVERSE-VIDEO [5]
REWIND
REWRITE
RD [2]
RF [2]
RH [2]
RIGHT
RIGHT-JUSTIFY [5]
ROLLBACK [1]
ROLLING [1]
ROUNDED
RUN
SAME
SCREEN [5]
SD
SEARCH
SECTION
SECURE [5]
SECURITY
SEGMENT [2]
SEGMENT-LIMIT
SELECT
SEND [2]
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SHARED [3]
SIGN
SIZE
SKIP1 [1]
SKIP2 [1]
SKIP3 [1]
SORT
SORT-MERGE
SORT-RETURN [4]
SOURCE [2]
SOURCE-COMPUTER
SPACE
SPACE-FILL [5]
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
STANDARD-2
START
STARTING [1]
STATUS
STOP
STORE [3]
STRING
SUB-QUEUE-1 [2]
SUB-QUEUE-2 [2]
SUB-QUEUE-3 [2]
SUB-SCHEMA [3]
SUBFILE [1]
SUBTRACT
SUM [2]
SUPPRESS
SYMBOLIC [2]
SYNC
SYNCHRONIZED
TABLE [2]
TALLYING
TAPE

Reserved Word
TENANT [3]
TERMINAL
TERMINATE [2]
TEST
TEXT [2]
THAN
THEN
THROUGH
THRU
TIME
TIMES
TITLE [1]
TO
TOP
TRAILING
TRAILING-SIGN [5]
TRANSACTION [1]
TRUE
TYPE [2]
UNDERLINE [5]
UNEQUAL [3]
UNIT
UNSTRING
UNTIL
UP
UPDATE [5]
UPON
USAGE
USAGE-MODE [3]
USE
USING
VALID [3]
VALIDATE [3]
VALUE
VALUES
VARYING
WAIT [3]
WHEN
WHEN-COMPILED [1]
WITH
WITHIN [3]
WORDS
WORKING-STORAGE
WRITE
ZERO
ZEROES
ZERO-FILL [5]
ZEROS
<
< =
+
*
**
-
/
>
> =
=

Appendix F. File Structure Support Summary and Status Key Values

Table 57 lists the required and optional entries for various types of file structures supported. Any file with a device type of disk can be assigned to a database or non-database auxiliary storage file. The codes used are as follows:

	Not applicable
B	Optional for a work station that supports subfiles
C	Optional entry, treated as comments only
D	Optional for file assigned to DATABASE-, not allowed if not assigned to a database file
I	Optional for a file opened for input or input-output
J	Optional for a file opened for input-output
O	Optional
R	Required
S	Required for a work station that supports subfiles
X	Required; syntax-checked, but treated as documentation.

Table 58 on page 504 and Table 59 on page 505 contain status key values and their meanings.

Table 57 (Page 1 of 4). File Structure Support

Device Type	Printer	Tape	Disk Seq	Disk Rel Seq	Disk Rel Random	Disk Rel Dynamic	Disk IDX Seq	Disk IDX Random	Disk IDX Dynamic	Work Station	Diskette	Format File
Environment Division												
RERUN...RECORDS	C	C	C	C	C	C	C	C	C	C	C	C
SAME	O	O	O	O	O	O	O	O	O	O	O	O
AREA	C	C	C	C	C	C	C	C	C	C	C	C
RECORD AREA	O	O	O	.	O	O	O	O	O	O	O	O
SORT AREA	.	C	C	.	.	.	.	.	.	.	.	.
SORT MERGE AREA	.	C	C	.	.	.	.	.	.	.	.	.
MULTIPLE FILE TAPE	.	C	.	.	.	.	.	.	.	.	.	.
COMMITMENT CONTROL	.	.	D	D	D	D	D	D	D	.	.	.
SELECT	R	R	R	R	R	R	R	R	R	R	R	R
ASSIGN	R	R	R	R	R	R	R	R	R	R	R	R
OPTIONAL	.	I	I	.	.	.	.	.	.	.	I	I
ORGANIZATION	O	O	O	R	R	R	R	R	R	R	O	O
SEQUENTIAL	O	O	O	.	.	.	.	.	.	.	O	O
RELATIVE	.	.	.	R	R	R	.	.	.	.	.	.
INDEXED	.	.	.	.	.	.	R	R	R	.	.	.
TRANSACTION	.	.	.	.	.	.	.	.	.	R	.	.

Table 57 (Page 2 of 4). File Structure Support

Device Type	Printer	Tape	Disk Seq	Disk Rel Seq	Disk Rel Random	Disk Rel Dynamic	Disk IDX Seq	Disk IDX Random	Disk IDX Dynamic	Work Station	Diskette	Format File
ACCESS	O	O	O	O	R	R	O	R	R	O	O	O
SEQUENTIAL	O	O	O	O	.	.	O	.	.	O	O	O
RANDOM	.	.	.	.	R	.	.	R	.	.	.	.
DYNAMIC	.	.	.	.	.	R	.	.	R	S	.	.
RESERVE	C	C	C	C	C	C	C	C	C	.	C	C
RELATIVE KEY	.	.	.	O	R	R	.	.	.	S	.	.
RECORD KEY	.	.	.	.	.	.	R	R	R	.	.	.
DUPLICATES	.	.	.	.	.	.	D	D	D	.	.	.
FILE STATUS	O	O	O	O	O	O	O	O	O	O	O	O
CONTROL-AREA	.	.	.	.	.	.	.	.	.	O	.	.
DATA DIVISION												
LABEL RECORDS	X	R	X	X	X	X	X	X	X	X	X	X
STANDARD	.	O	R	R	R	R	R	R	R	O	R	R
OMITTED	R	O	.	.	.	.	.	.	.	O	.	.
VALUE OF	C	C	C	C	C	C	C	C	C	C	C	C
BLOCK CONTAINS	O	O	O	O	O	O	O	O	O	O	O	O
RECORD CONTAINS	O	O	O	O	O	O	O	O	O	O	O	O
DATA RECORDS	O	O	O	O	O	O	O	O	O	O	O	O
CODE-SET	.	O	.	.	.	.	.	.	.	.	O	.
LINEAGE	O	.	.	.	.	.	.	.	.	.	.	.
PROCEDURE DIVISION												
OPEN	R	R	R	R	R	R	R	R	R	R	R	R
INPUT	.	O	O	O	O	O	O	O	O	.	O	O
OUTPUT	R	O	O	O	O	O	O	O	O	.	O	O
I-O	.	.	O	O	O	O	O	O	O	R	.	O
NO REWIND	.	I	.	.	.	.	.	.	.	.	.	.
REVERSED	.	I	.	.	.	.	.	.	.	.	.	.
EXTEND	.	O	O	.	.	.	.	.	.	.	.	O
CLOSE	R	R	R	R	R	R	R	R	R	R	R	R
REEL/UNIT	.	O	.	.	.	.	.	.	.	.	.	.
REMOVAL	.	O	.	.	.	.	.	.	.	.	.	.
NO REWIND	.	O	.	.	.	.	.	.	.	.	.	.
NO REWIND	.	O	.	.	.	.	.	.	.	.	.	.
WITH LOCK	O	O	O	O	O	O	O	O	O	O	O	O
READ	.	I	I	I	I	I	I	I	I	I	I	I
NEXT	.	.	.	.	.	I	.	.	I	.	.	.
FIRST	.	.	.	.	.	.	.	.	D	.	.	.
LAST	.	.	.	.	.	.	.	.	D	.	.	.
PRIOR	.	.	.	.	.	.	.	.	D	.	.	.

Table 57 (Page 3 of 4). File Structure Support

Device Type	Printer	Tape	Disk Seq	Disk Rel Seq	Disk Rel Random	Disk Rel Dynamic	Disk IDX Seq	Disk IDX Random	Disk IDX Dynamic	Work Station	Diskette	Format File
INTO	.	I	I	I	I	I	I	I	I	I	I	I
WITH NO LOCK	.	.	J	J	J	J	J	J	J	.	.	.
KEY IS	.	.	.	.	.	.	.	I	I	.	.	.
AT END	.	I	I	I	.	I	I	.	I	I	I	I
NOT AT END	.	I	I	I	.	I	I	.	I	I	I	I
INVALID KEY	.	.	.	.	I	I	.	I	I	B	.	.
NOT INVALID KEY	.	.	.	.	I	I	.	I	I	B	.	.
FORMAT	.	.	D	.	.	.	D	D	D	I	.	R
NEXT MODIFIED	.	.	.	.	.	.	.	.	.	B	.	.
SUBFILE	.	.	.	.	.	.	.	.	.	B	.	.
INDICATORS	.	.	.	.	.	.	.	.	.	I	.	.
TERMINAL	.	.	.	.	.	.	.	.	.	O	.	.
NO DATA	.	.	.	.	.	.	.	.	.	O	.	.
WRITE	O	O	O	O	O	O	O	O	O	O	O	O
FROM	O	O	O	O	O	O	O	O	O	O	O	O
INVALID KEY	.	.	.	O	O	O	O	O	O	B	.	.
NOT INVALID KEY	.	.	.	O	O	O	O	O	O	B	.	.
ADVANCING	O	.	.	.	.	.	.	.	.	.	.	.
AT END-OF-PAGE	O	.	.	.	.	.	.	.	.	.	.	.
NOT AT END-OF-PAGE	O	.	.	.	.	.	.	.	.	.	.	.
FORMAT	.	.	D	.	.	.	D	D	D	R	.	R
STARTING	.	.	.	.	.	.	.	.	.	O	.	.
ROLLING	.	.	.	.	.	.	.	.	.	O	.	.
INDICATORS	.	.	.	.	.	.	.	.	.	O	.	.
SUBFILE	.	.	.	.	.	.	.	.	.	B	.	.
TERMINAL	.	.	.	.	.	.	.	.	.	O	.	.
START	.	.	.	O	.	O	O	.	O	.	.	.
KEY	.	.	.	O	.	O	O	.	O	.	.	.
INVALID KEY	.	.	.	O	.	O	O	.	O	.	.	.
NOT INVALID KEY	.	.	.	O	.	O	O	.	O	.	.	.
FORMAT	.	.	.	.	.	.	D	D	D	.	.	.
REWRITE	.	.	O	O	O	O	O	O	O	B	.	.
FROM	.	.	O	O	O	O	O	O	O	B	.	.
INVALID KEY	.	.	.	.	O	O	.	O	O	B	.	.
NOT INVALID KEY	.	.	.	.	O	O	.	O	O	B	.	.
FORMAT	.	.	.	.	.	.	.	D	D	B	.	.
INDICATORS	.	.	.	.	.	.	.	.	.	B	.	.
SUBFILE	.	.	.	.	.	.	.	.	.	S	.	.
TERMINAL	.	.	.	.	.	.	.	.	.	O	.	.

Table 57 (Page 4 of 4). File Structure Support

Device Type	Printer	Tape	Disk Seq	Disk Rel Seq	Disk Rel Random	Disk Rel Dynamic	Disk IDX Seq	Disk IDX Random	Disk IDX Dynamic	Work Station	Diskette	Format File
DELETE	.	.	.	O	O	O	O	O	O	.	.	.
INVALID KEY	.	.	.	.	O	O	.	O	O	.	.	.
NOT INVALID KEY	.	.	.	.	O	O	.	O	O	.	.	.
FORMAT	.	.	.	.	.	.	.	D	D	.	.	.
USE	O	O	O	O	O	O	O	O	O	O	O	O
EXCEPTION/ERROR	O	O	O	O	O	O	O	O	O	O	O	O
FOR DEBUGGING	O	O	O	O	O	O	O	O	O	O	O	O
COMMIT	.	.	D	D	D	D	D	D	D	.	.	.
ROLLBACK	.	.	D	D	D	D	D	D	D	.	.	.
ACQUIRE	.	.	.	.	.	.	.	.	.	O	.	.
DROP	.	.	.	.	.	.	.	.	.	O	.	.

Return codes are set by the system after transaction I-O, which involves ICF files or DISPLAY files.

For more information about return codes, see the *COBOL/400 User's Guide*.

Table 58. File Status Keys and their Corresponding Return Codes

File Status Key	Major Return Code	Minor Return Code	Explanation
00	00 03 08 09	XX XX (except 09) 00 00	Normal completion (operation was successful). No data received. Acquire operation attempted to acquire an already active session or device. File has been dynamically created for OPEN OUTPUT. (See the description for GENOPT(*CRTF) in the *COBOL/400 User's Guide* for further information about dynamic file creation.)
10	11	00	Read-from-invited-program-device rejected; no invites outstanding.
30	80	XX	Permanent system error. The session has been ended.
92	81	XX	Permanent device or session error.
9A	02 03	XX 09	Job being cancelled (controlled).
9C	82	XX	Open or acquire failed; session was not started.
9G	34	XX	Output exception to device or session.
9I	04	XX	Output exception to device or session.
9K	83	E0	Format not found.
9N	83	XX (except E0)	Session error. Session is still active.

File Status Key Values and Meanings

For information about **error handling**, refer to "COBOL/400 Error Handling" in the *COBOL/400 User's Guide*.

Table 59 (Page 1 of 5). File Status Key Values

High Order Digit	Meaning	Low Order Digit	Meaning
0	Successful Completion	0	No further information
		2	The READ statement was successfully executed, but a duplicate key was detected. That is, the key value for the current key of reference was equal to the value of the key in the next record. For information about enabling file status 02 see Table 44 on page 357 and the accompanying notes under the READ statement.
		4	A READ statement was successfully executed, but the length of the record being processed did not conform to the fixed file attributes for that file.
		5	An OPEN statement is successfully executed, but the referenced optional file is not present at the time the OPEN statement is executed. If the open mode is I-O or EXTEND, the file has been created.
		7	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 was on a non-reel/unit medium.
		M	Last record written to a subfile. CPF5003
		P	The file has been opened successfully, but it contains null-capable fields.
		Q	A CLOSE statement for a sequentially-processed relative file was successfully executed. The file was created with the *INZDLT and *NOMAX options, so its boundary has been set to the number of records written.
1	At end conditions	0	A sequential READ statement was attempted and no next logical record existed in the file because the end of the file had been reached (no invites outstanding) CPF4740, CPF5001, CPF5025.
		2	IBM Extension No modified subfile record found. CPF5037 End of IBM Extension
		4	A sequential READ statement was attempted for a relative file and the number of significant digits in the relative record number was larger than the size of the relative key data item described for the file.

Table 59 (Page 2 of 5). File Status Key Values

High Order Digit	Meaning	Low Order Digit	Meaning
2	Invalid Key	1	A sequence error exists for a sequentially accessed indexed file. The prime record key value has been changed by the program between the successful execution of a READ statement and the execution of the next REWRITE statement for that file, or the ascending requirements for successive record key values were violated. Alternatively, the program has changed the record key value between a successful READ and subsequent REWRITE or DELETE operation on a randomly or dynamically-accessed file with duplicate keys.
		2	An attempt was made to write a record that would create a duplicate key in a relative file; or an attempt was made to write or rewrite a record that would create a duplicate prime record key in an indexed file. CPF4759, CPF5008, CPF5026, CPF5034, CPF5084, CPF5085.
		3	An attempt was made to randomly access a record that does not exist in the file. CPF5001, CPF5006, CPF5013, CPF5020, CPF5025.
		4	An attempt was made to write beyond the externally defined boundaries of a relative or indexed file. Or, a sequential WRITE statement was attempted for a relative file and the number of significant digits in the relative record number was larger than the size of the relative record key data item described for the file. CPF5006, CPF5018, CPF5021, CPF5043, CPF5272.
3	Permanent error condition	0	No further information CPF4192, CPF5101, CPF5102, CPF5129, CPF5030, CPF5143.
		4	A permanent error exists because of a boundary violation; an attempt was made to write beyond the externally-defined boundaries of a sequential file. CPF5116, CPF5018, CPF5272 if organization is sequential.
		5	An OPEN statement with the INPUT, I-O, or EXTEND phrase was attempted on a non-optional file that was not present. CPF4101, CPF4102, CPF4103, CPF4207, CPF9812.
		7	An OPEN statement was attempted on a file that would not support the open mode specified in the OPEN statement. Possible violations are: 1. The EXTEND or OUTPUT phrase was specified but the file would not support write operations. 2. The I-O phrase was specified but the file would not support the input and output operations permitted. 3. The INPUT phrase was specified but the file would not support read operations. CPF4194.
		8	An OPEN statement was attempted on a file previously closed with lock.
		9	The OPEN statement was unsuccessful because a conflict was detected between the fixed file attributes and the attributes specified for that file in the program. Level check error. CPF4131.

Table 59 (Page 3 of 5). File Status Key Values

High Order Digit	Meaning	Low Order Digit	Meaning
4	Logic error condition	1	An OPEN statement was attempted for a file in the open mode.
		2	A CLOSE statement was attempted for a file not in the open mode.
		3	For a sequential file in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of a REWRITE statement was not a successfully executed READ statement. For relative and indexed files in the sequential access mode, the last input-output statement executed for the file prior to the execution of a DELETE or REWRITE statement was not a successfully executed READ statement.
		4	A boundary violation exists because an attempt was made to rewrite a record to a file and the record was not the same size as the record being replaced.
		6	A sequential READ statement was attempted on a file open in the input or I-O mode and no valid next record had been established because the preceding START statement was unsuccessful, or the preceding READ statement was unsuccessful or caused an at end condition. CPF5001, CPF5025, CPF5183.
		7	The execution of a READ or START statement was attempted on a file not open in the input or I-O mode.
		8	The execution of a WRITE statement was attempted on a sequential file not open in the output, or extend mode. The execution of a WRITE statement was attempted on an indexed or relative file not open in the I-O, output, or extend mode.
		9	The execution of a DELETE or REWRITE statement was attempted on a file not open in the I-O mode.

Table 59 (Page 4 of 5). File Status Key Values

High Order Digit	Meaning	Low Order Digit	Meaning
9	Other Errors	0	Other errors: • File not found • Member not found • Unexpected I-O exceptions CPF4101, CPF4102, CPF4103 if a USE is applicable for the file (on OPEN OUTPUT, non-optional file). The following exceptions are monitored generically: CPF4101 through CPF4399 CPF4501 through CPF4699 CPF4701 through CPF4899 CPF4901 through CPF4999 CPF5001 through CPF5099 CPF5101 through CPF5399 CPF5501 through CPF5699 These exceptions are caught, and FILE STATUS is set to 90. **With Standard Error Handling**: If there is an applicable file status clause (but not an applicable USE procedure), the file status is updated, and control returns to the program. In the absence of a file status clause, USE procedure, AT END clause, or INVALID KEY clause to handle the error, a run-time message is issued, giving you the option to end or return to the program. **Without Standard Error Handling**: If a USE procedure applies, it runs. Otherwise, the program ends and gives the operator the exception and the option to cancel, take a partial dump, or take a full dump.
		1	Undefined or unauthorized access type CPF2207, CPF4104, CPF4236, CPF5057, CPF5109, CPF5134, CPF5279.
9	Other errors	2	Logic error: • File locked • File already open • I-O to closed file • READ after end of file • CLOSE on unopened file CPF4106, CPF4132, CPF4740, CPF5013, CPF5067, CPF5070, CPF5119, CPF5145, CPF5146, CPF5149, CPF5176, CPF5209.
		4	No file position indicator REWRITE/DELETE when ***not*** sequential access, and last operation was not a successful READ.
		5	Invalid or incomplete file information (1) Duplicate keys specified in COBOL program. The file has been successfully opened, but indexed database file created with unique key; or (2) Duplicate keys not specified in COBOL program, and indexed database file created allowing duplicate keys.
		9	Undefined (display or ICF).

Table 59 (Page 5 of 5). File Status Key Values

High Order Digit	Meaning	Low Order Digit	Meaning
9	Other Errors	A	Job ended in a controlled manner by CL command ENDJOB, PWRDWNSYS, ENDSYS, or ENDSBS CPF4741. Escape message sent during an accept input operation, READ from invited program device (multiple device listings only).
		C	Acquire failed: session was not started.
		D	Record is locked CPF5027, CPF5032.
		G	Output exception to device or session.
		H	ACQUIRE operation failed. Resource owned by another program, or unavailable. (9H is the result when an ACQUIRE operation causes any of the OS/400 exceptions monitored for 90, or 9N to occur.)
		I	WRITE operation failed CPF4702, CPF4737, CPF5052, CPF5076.
		K	Invalid format-name: format not found CPF5022, CPF5023, CPF5053, CPF5054, CPF5121, CPF5152, CPF5153, CPF5186, CPF5187.
		M	Last record written to subfile. CPF5003. To convert file status 9M to 0M, use PROCESS statement option FS9MTO0M.
		N	Temporary (potentially recoverable) hardware I-O error. (Error during communication session.) CPF4145, CPF4146, CPF4193, CPF4229, CPF4291, CPF4299, CPF4354, CPF4526, CPF4542, CPF4577, CPF4592, CPF4602, CPF4603, CPF4611, CPF4612, CPF4616, CPF4617, CPF4622, CPF4623, CPF4624, CPF4625, CPF4628, CPF4629, CPF4630, CPF4631, CPF4632, CPF4705, CPF5107, CPF5128, CPF5166, CPF5198, CPF5280, CPF5282, CPF5287, CPF5293, CPF5352, CPF5353, CPF5517, CPF5524, CPF5529, CPF5530, CPF5532, CPF5533.
		P	OPEN failed because file cannot be placed under commitment control CPF4293, CPF4326, CPF4327, CPF4328, CPF4329.
		Q	An OPEN statement for a randomly- or dynamically-accessed relative file failed because its size was *NOMAX. Change the file size (for example, using CHGPF) to the size you expect, and submit the program again.
		S	REWRITE or DELETE failed because last READ operation specified NO LOCK.
		U	Cannot complete READ PRIOR because records are left in block from READ NEXT, or vice versa. CPF5184. Close the file, then open it again.

Attribute Data Formats

The layouts and values of the attribute data are system dependent. The following formats are for the COBOL/400 language.

```
 01  DISPLAY-ICF-ATTRIBUTES.
     02  PROGRAM-DEVICE-NAME        PIC X(10).
     02  DEVICE-DESCRIPTION-NAME    PIC X(10).
     02  USER-ID                    PIC X(10).
     02  DEVICE-CLASS               PIC X.
*          D - DISPLAY
*          I - ICF
*          U - UNKNOWN
     02  DEVICE-TYPE                PIC X(6).
*          '      ' - UNKNOWN
*          '3179  ' - 3179 DISPLAY
*          '317902' - 3179 MOD 2 DISPLAY
*          '3180  ' - 3180 DISPLAY
*          '3196A ' - 3196 MOD A1/A2 DISPLAY
*          '3196B ' - 3196 MOD B1/B2 DISPLAY
*          '3197C1' - 3197 MOD C1 DISPLAY
*          '3197C2' - 3197 MOD C2 DISPLAY
*          '3197D1' - 3197 MOD D1 DISPLAY
*          '3197D2' - 3197 MOD D2 DISPLAY
*          '3197W1' - 3197 MOD W1 DISPLAY
*          '3197W2' - 3197 MOD W2 DISPLAY
*          '3270  ' - 3270 DISPLAY
*          '3476EA' - 3476 MOD EA DISPLAY
*          '3476EC' - 3476 MOD EC DISPLAY
*          '3477FG' - 3477 MOD FG DISPLAY
*          '3477FA' - 3477 MOD FA DISPLAY
*          '3477FC' - 3477 MOD FC DISPLAY
*          '3477FD' - 3477 MOD FD DISPLAY
*          '3477FW' - 3477 MOD FW DISPLAY
*          '3477FE' - 3477 MOD FE DISPLAY
*          '525111' - 5251 DISPLAY
*          '5291  ' - 5291 DISPLAY
*          '5292  ' - 5292 DISPLAY
*          '529202' - 5292 MOD 2 DISPLAY
*          '5555B1' - 5555 MOD B01 DISPLAY
*          '5555C1' - 5555 MOD C01 DISPLAY
*          '5555E1' - 5555 MOD E01 DISPLAY
*          '5555F1' - 5555 MOD F01 DISPLAY
*          '5555G1' - 5555 MOD G01 DISPLAY
*          '5555G2' - 5555 MOD G02 DISPLAY
*          'DHCF77' - 3277 DHCF DISPLAY
*          'DHCF78' - 3278 DHCF DISPLAY
*          'DHCF79' - 3279 DHCF DISPLAY
*          'APPC  ' - ADVANCED-PROGRAM-TO-PROGRAM COMMUNICATIONS DEVICE
*          'ASYNC ' - ASYNCHRONOUS COMMUNICATION DEVICE
*          'BSC   ' - BISYNCHRONOUS COMMUNICATION
*          'BSCEL ' - BSCEL COMMUNICATION DEVICE
*          'FINANC' - ICF FINANCE COMMUNICATION DEVICE
```

```
*          'INTRA ' - INTRA SYSTEMS COMMUNICATION
*          'LU1   ' - LU1 COMMUNICATION DEVICE
*          'RETAIL' - RETAIL COMMUNICATION DEVICE
*          'SNUF  ' - SNA UPLINE FACILITY COMMUNICATION DEVICE
*          'NVT   ' - NETWORK VIRTUAL TERMINAL (NVT)
     02  REQUESTOR-DEVICE           PIC X.
*          N - NOT A REQUESTOR DEVICE
*          Y - A REQUESTOR DEVICE
     02  ACQUIRE-STATUS             PIC X.
*          N - DEVICE NOT ACQUIRED
*          Y - DEVICE ACQUIRED
     02  INVITE-STATUS              PIC X.
*          N - DEVICE NOT INVITED
*          Y - DEVICE INVITED
     02  DATA-AVAILABLE-STATUS      PIC X.
*          N - NO DATA IS AVAILABLE
*          Y - INVITED DATA AVAILABLE
     02  DISPLAY-DIMENSIONS.
         03  NUMBER-OF-ROWS         PIC S9(4)  COMP-4.
         03  NUMBER-OF-COLUMNS      PIC S9(4)  COMP-4.
     02  DISPLAY-ALLOW-BLINK        PIC X.
*          N - NOT BLINK CAPABLE
*          Y - BLINK CAPABLE
     02  ONLINE-OFFLINE-STATUS      PIC X.
*          O - DISPLAY IS ONLINE
*          F - DISPLAY IS OFFLINE
     02  DISPLAY-LOCATION           PIC X.
*          L - LOCAL DISPLAY
*          R - REMOTE DISPLAY
     02  DISPLAY-TYPE               PIC X.
*          A - ALPHANUMERIC OR KATAKANA
*          I - DBCS
     02  KEYBOARD-TYPE              PIC X.
*          A - ALPHANUMERIC OR KATAKANA KEYBOARD
*          I - DBCS KEYBOARD
     02  CONVERSATION-STATUS        PIC X.
*          N - CONVERSATION NOT INITIATED
*          Y - CONVERSATION INITIATED
*          (VALID FOR ALL COMMUNICATION TYPES).
     02  SYNCHRONIZATION-LEVEL      PIC X.
*          0 - SYNCHRONIZATION LEVEL 0  (SYNLVL(*NONE))
*          1 - SYNCHRONIZATION LEVEL 1  (SYNLVL(*CONFIRM))
*          (APPC APPLICATIONS ONLY)
     02  CONVERSATION-USED          PIC X.
*          M - MAPPED CONVERSATION
*          B - BASIC CONVERSATION
*          (APPC APPLICATIONS ONLY)
     02  REMOTE-LOCATION-NAME       PIC X(8).
*          (ALL COMMUNICATION TYPES)
     02  LOCAL-LU-NAME              PIC X(8).
*          (APPC APPLICATIONS ONLY)
     02  LOCAL-NETWORK-ID           PIC X(8).
*          (APPC APPLICATIONS ONLY)
```

```
           02  REMOTE-LU-NAME              PIC X(8)
      *          (APPC APPLICATIONS ONLY)
           02  REMOTE-NETWORK-ID           PIC X(8).
      *          (APPC APPLICATIONS ONLY)
           02  MODE                        PIC X(8).
      *          (APPC APPLICATIONS ONLY)
           02  FILLER                      PIC X(43).
      *          RESERVED
           02  CALLING-PARTY-ID.
               03  REMOTE-NUMBER-LENGTH        PIC S9(4)    COMP-4.
               03  REMOTE-NUMBERING-TYPE       PIC X(2).
      *              00 - UNKNOWN
      *              01 - INTERNATIONAL
      *              02 - NATIONAL
      *              03 - NETWORK-SPECIFIC
      *              04 - SUBSCRIBER
      *              06 - ABBREVIATED
      *              07 - RESERVED
               03  REMOTE-NUMBERING-PLAN       PIC X(2).
      *              00 - UNKNOWN
      *              01 - ISDN/TELEPHONY
      *              03 - DATA
      *              04 - TELEX
      *              08 - NATIONAL STANDARD
      *              09 - PRIVATE
      *              15 - RESERVED
               03  REMOTE-NUMBER               PIC X(40).
               03  FILLER                      PIC X(4).
      *              RESERVED
               03  REMOTE-SUBADDR-LENGTH       PIC S9(4)    COMP-4.
               03  REMOTE-SUBADDR-TYPE         PIC X(2).
      *              00 - NSAP
      *              02 - USER SPECIFIED
               03  REMOTE-SUBADDRESS           PIC X(40).
               03  FILLER                      PIC X.
      *              RESERVED
               03  CALL-TYPE                   PIC X.
      *              0 - CALL IN
      *              1 - CALL OUT
      *              2 - NON-ISDN
               03  REMOTE-NETADDR-LENGTH       PIC S9(4)    COMP-4.
               03  REMOTE-NETADDRESS           PIC X(32).
               03  FILLER                      PIC X(4).
      *              RESERVED
               03  REMOTE-ADDREXT-LENGTH       PIC S9(4)    COMP-4.
               03  REMOTE-ADDREXT-TYPE         PIC X.
      *              0 - ISO 8348/AD2
      *              2 - NOT ISO 8348/AD2
               03  REMOTE-ADDRESS-EXTENSION    PIC X(40).
               03  FILLER                      PIC X(4).
      *              RESERVED
               03  X25-CALL-TYPE               PIC X.
      *              0 - INCOMING SVC
      *              1 - OUTGOING SVC
      *              2 - NOT X25 SVC
               03  TRANSACTION-PROGRAM-NAME    PIC X(64).
```

OPEN-FEEDBACK and I-O-FEEDBACK Data Areas

OPEN-FEEDBACK

The OPEN-FEEDBACK area is part of the open data path (ODP) that contains information about the OPEN operation. This information is set during OPEN processing and is available as long as the file is open.

This area provides information about the file that the program is using. It contains:

- Information about the file that is currently open, such as:
  - File name
  - File type.
- Information that depends on the type of file that is opened, such as:
  - Printer size
  - Screen size
  - Diskette or tape labels.

I-O-FEEDBACK

The system updates the I-O-FEEDBACK area each time a block transfers between the operating system and the program. A block can contain one or more records.

The I-O-FEEDBACK area is not updated after each read or write operation for files in which multiple records are blocked and unblocked by COBOL. If the I-O-FEEDBACK information is needed after each read or write operation in the program, the user can do either of the following:

- Prevent the compiler from generating blocking and unblocking code by not satisfying one of the conditions listed under "Unblocking Input Records and Blocking Output Records" in the *COBOL/400 User's Guide*.
- Specify SEQONLY(*NO) on the Override with database file (OVRDBF) CL command.

Preventing the compiler from generating blocking and unblocking code is more efficient than specifying SEQONLY(*NO).

Even when the compiler generates blocking and unblocking code, certain OS/400 restrictions can cause blocking and unblocking to not be processed. In these cases, a performance improvement will not be realized. However, the I-O-FEEDBACK area will be updated after each read or write operation.

The I-O-FEEDBACK area contains information about the I-O operation. This area consists of a common area and a device-dependent area. The device-dependent area varies in length and content depending on the file type. This area follows the I-O-FEEDBACK common area and can be obtained by specifying the receiving identifier large enough to include the common area and the appropriate device-dependent area.

The I-O-FEEDBACK area contains information about the last I-O operation, such as:

- Device name
- Device type
- AID character
- Error information for some devices.

See the *Data Management Guide* for a layout and description of the data areas contained in the OPEN-FEEDBACK and I-O-FEEDBACK areas.

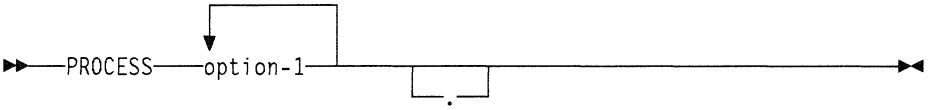
Appendix G. PROCESS Statement, PICTURE Symbols, ASSIGN Clause

PROCESS Statement

The PROCESS statement is an optional part of the COBOL source program. It lets you specify options that you would normally specify at compilation time. Options specified in the PROCESS statement override the corresponding options specified in the CRTCBLPGM CL command.

The format of the PROCESS statement is as follows:

Format



The following tables indicate the allowable PROCESS statement options and the equivalent CRTCBLPGM command parameters and options. Defaults are underlined.

Descriptions of the PROCESS statement options correspond to the parameter and option descriptions for the CRTCBLPGM parameter. See the *COBOL/400 User's Guide* for more information.

PROCESS Statement Option	CRTCBLPGM
	GENLVL Parameter Option
GENLVL(nn)	nn

PROCESS Statement Options	CRTCBLPGM
	OPTION Parameter Options
GEN NOGEN	*GEN *NOGEN
NOMAP MAP	*NOMAP *MAP
NONUMBER NUMBER LINENUMBER	*NONUMBER *NUMBER *LINENUMBER
NOSECLVL SECLVL	*NOSECLVL *SECLVL
NOOPTIONS OPTIONS	*NOOPTIONS *OPTIONS
QUOTE APOST	*QUOTE *APOST
NOSEQUENCE SEQUENCE	*NOSEQUENCE *SEQUENCE
SOURCE (or SRC) NOSOURCE (or NOSRC)	*SOURCE (or *SRC) *NOSOURCE (or *NOSRC)
NOVBSUM VBSUM	*NOVBSUM *VBSUM
NOXREF XREF	*NOXREF *XREF
PRTCORR NOPRTCORR	*PRTCORR *NOPRTCORR

PROCESS Statement Options	CRTCBLPGM
	GENOPT Parameter Options
NOINZDLT INZDLT	*NOINZDLT *INZDLT
NOLIST LIST	*NOLIST *LIST
STDERR NOSTDERR	*STDERR *NOSTDERR
NODDSFILLER DDSFILLER	*NODDSFILLER *DDSFILLER
NOSYNC SYNC	*NOSYNC *SYNC
NOCRTF CRTF	*NOCRTF *CRTF
NODUPKEYCHK DUPKEYCHK	*NODUPKEYCHK *DUPKEYCHK
NOEXTACCDSP EXTACCDSP	*NOEXTACCDSP *EXTACCDSP
NOBLK BLK	*NOBLK *BLK
STDINZ NOSTDINZ	*STDINZ *NOSTDINZ
FS21DUPKEY NOFS21DUPKEY	*FS21DUPKY *NOFS21DUPKY
RANGE NORANGE	*RANGE *NORANGE
UNREF NOUNREF	*UNREF *NOUNREF

PROCESS Statement Options	CRTCBLPGM
	CVTOPT Parameter Options
NOVARCHAR VARCHAR	*NOVARCHAR *VARCHAR
NODATETIME DATETIME	*NODATETIME *DATETIME
NOCVTGRAPHIC CVTGRAPHIC	*NOGRAPHIC *GRAPHIC

PROCESS Statement Options	CRTCBLPGM
	FLAGSTD Parameter Options
NOFIPS MINIMUM INTERMEDIATE HIGH	*NOFIPS *MINIMUM *INTERMEDIATE *HIGH
NOSEG SEG1 SEG2	*NOSEG *SEG1 *SEG2
NODEB DEB1 DEB2	*NODEB *DEB1 *DEB2
NOOBSOLETE OBSOLETE	*NOOBSOLETE *OBSOLETE

PROCESS Statement Options EXTDSPOPT(*a b c*)	CRTCBLPGM
	EXTDSPOPT Parameter Options
DFRWRT NODFRWRT	*DFRWRT *NODFRWRT
UNDSPCHR NOUNDSPCHR	*UNDSPCHR *NOUNDSPCHR
ACCUPDALL ACCUPDNE	*ACCUPDALL *ACCUPDNE

PROCESS Statement Options	CRTCBLPGM
	SAAFLAG Parameter Options
NOSAAFLAG SAAFLAG	*NOFLAG *FLAG

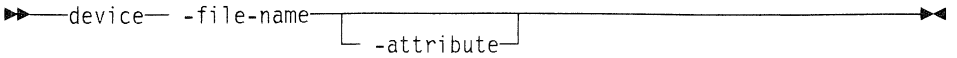
PROCESS Statement Option	CRTCBLPGM
	FLAG Parameter Option
FLAG(nn)	nn

PROCESS Statement Options	CRTCBLPGM
NOFS9MTO0M FS9MTO0M	not applicable
NOGRAPHIC GRAPHIC	not applicable

Symbols Used in the PICTURE Clause

Symbol	Meaning
A	Alphabetic character or space
B	Space insertion character
P	Decimal scaling position (not counted in size of data item)
S	Operational sign (not counted in size of data item unless a SIGN clause with optional SEPARATE CHARACTER phrase is specified)
V	Assumed decimal point (not counted in size of data item)
X	Alphanumeric character (any from the EBCDIC set)
Z	Zero suppression character
9	Numeric character
	IBM Extension
1	Boolean character
	End of IBM Extension
0	Zero insertion character
/	Slash insertion character
,	Comma insertion character
.	Decimal point or period editing control character
+	Plus sign insertion editing control character
-	Minus sign editing control character
CR	Credit editing control character
DB	Debit editing control character
*	Check protect insertion character
$	Currency symbol insertion character ($ is default).

Assignment-Names in the ASSIGN Clause



device:	PRINTER TAPEFILE DISKETTE DISK DATABASE WORKSTATION FORMATFILE
file-name:	1-10 character name
attribute:	SI (for separate indicator area)

Appendix H. SAA COBOL Compiler Options

The compiler options required for Systems Application Architecture programs are system-dependent.

Under the OS/400 operating system, the required compiler options for SAA processing using the CRTCBLPGM command are:

OPTION(*QUOTE *NOSEQUENCE *NONUMBER)

GENOPT(*CRTF *DUPKEYCHK *SYNC)

SAAFLAG(*FLAG)

For the PROCESS statement, the required compiler options for SAA processing are:

QUOTE, NOSEQUENCE, NONUMBER, CRTF, DUPKEYCHK, SYNC, SAAFLAG.

Appendix I. ACCEPT/DISPLAY and COBOL/2 Considerations

The COBOL/400 extended ACCEPT and DISPLAY statements are similar to the IBM COBOL/2 ACCEPT and DISPLAY statements (Format 2) with the following exceptions:

- Some WITH phrases are only syntax checked (as shown in the extended ACCEPT and DISPLAY syntax diagrams).
- ON ESCAPE is not used as an alternative to ON EXCEPTION.
- If phrases are duplicated in a displayed or an accepted data item, the COBOL/400 compiler issues a severe error message. The COBOL/2 compiler permits some duplication of phrases, such as UPON and BELL.
- AUTO-SKIP may be specified with a group item on a COBOL/400 extended ACCEPT statement but the COBOL/2 compiler generates a severe error message.
- BELL may be specified with a group item on a COBOL/400 extended ACCEPT statement but the COBOL/2 compiler generates a severe error message.
- The COBOL/400 compiler accepts and applies the following to the appropriate fields if they are specified with a group item. The COBOL/2 compiler generates a severe error message.

  REQUIRED
  SECURE
  LEFT-JUSTIFY
  RIGHT-JUSTIFY
  SPACE-FILL
  TRAILING-SIGN
  UPDATE
  ZERO-FILL

- The COBOL/400 extended DISPLAY statement handles only fixed-size tables. The COBOL/2 DISPLAY statement handles fixed and variable size tables.
- The COBOL/2 compiler justifies the signed numeric data (displayed and accepted) to the left, and the COBOL/400 compiler justifies these data items to the right.
- The COBOL/2 compiler handles special effects with figurative constants when found in the DISPLAY statement (for example, DISPLAY SPACE will do the same as DISPLAY WITH BLANK SCREEN), while the COBOL/400 compiler does not apply any special effects to the figurative constants when found as data items to be displayed in the extended DISPLAY statement.
- The COBOL/2 compiler uses all of the screen positions for displayable data items, while the COBOL/400 compiler always needs one byte preceding each displayable data item for the attribute byte. For this reason, line 1 and column 1 cannot be used on the COBOL/400 extended ACCEPT or DISPLAY statement. (Error message LBE7208 is issued.)
- When one ACCEPT or DISPLAY statement contains the UNDERLINE, HIGHLIGHT and REVERSE-VIDEO phrases in one WITH phrase, the HIGHLIGHT phrase is ignored. A warning message (LBL0265) is generated at compile time if this combination is coded. In an extended DISPLAY statement, the UPON CRT-UNDER phrase is equivalent to the UNDERLINE phrase. To protect a field from being displayed on the screen, use the SECURE option.

- Unless you specify the *NODFRWRT option of the CRTCBLPGM EXTDSPOPT parameter, the COBOL/400 compiler buffers all extended DISPLAY statements until the next ACCEPT statement is encountered.
- Under the *NOUNDSPCHR compiler option, values below hexadecimal 40 cause undesirable results in extended ACCEPT and extended DISPLAY operations. To overcome this hardware limitation, use the (default) *UNDSPCHR option.
- The COBOL/400 compiler does not provide run-time configuration options.
- The length of the data-name in the CRT STATUS clause on the COBOL/2 compiler is 3 bytes, and the length on the COBOL/400 compiler is 6 bytes.

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Readers' Comments

Application System/400
Languages:
Systems Application Architecture AD/Cycle
COBOL/400 Reference
Version 2

Publication No. SC09-1380-01

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments
SC09-1380-01

IBM®

Cut
Alo

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 Don Mills Road
North York, ON Canada M3C 1V7

Fold and Tape | **Please do not staple** | Fold and Tape

Cut
Alc

Readers' Comments

Application System/400
Languages:
Systems Application Architecture AD/Cycle
COBOL/400 Reference
Version 2

Publication No. SC09-1380-01

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments
SC09-1380-01

Cut
Alo

Fold and Tape **Please do not staple** Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 Don Mills Road
North York, ON Canada M3C 1V7

Fold and Tape **Please do not staple** Fold and Tape

Cut
Alo

Readers' Comments

Application System/400
Languages:
Systems Application Architecture AD/Cycle
COBOL/400 Reference
Version 2

Publication No. SC09-1380-01

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments
SC09-1380-01

IBM®

Cut
Alo

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 Don Mills Road
North York, ON Canada M3C 1V7

Fold and Tape | **Please do not staple** | Fold and Tape

Cut
Alo

Readers' Comments

Application System/400
Languages:
Systems Application Architecture AD/Cycle
COBOL/400 Reference
Version 2

Publication No. SC09-1380-01

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Phone No.

Readers' Comments
SC09-1380-01

Cut
Alo

Fold and Tape **Please do not staple** Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 Don Mills Road
North York, ON Canada M3C 1V7

Fold and Tape **Please do not staple** Fold and Tape

Cu
Al

Program Number: 5738-CB1

Printed in Ireland by Printech International plc

SC09-1380-01