IBM i
Version 7.2

Programming
IBM Rational Development Studio for i
ILE COBOL Language Reference

SC09-2539-08

Note

Before using this information and the product it supports, read the information in "Notices" on page 597.

This edition applies to IBM® Rational® Development Studio for i (product number 5770-WDS) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

This document may contain references to Licensed Internal Code. Licensed Internal Code is Machine Code and is licensed to you under the terms of the IBM License Agreement for Machine Code.

Contents

Part 1. ILE COBOL Language Reference

This reference describes the Integrated Language Environment® COBOL (ILE COBOL) programming language. It provides information on the structure of the ILE COBOL programming language and the structure of an ILE COBOL source program. It also provides a description of all Identification Division paragraphs, Environment Division clauses, Data Division clauses, Procedure Division statements, and Compiler-Directing statements.

This book refers to other IBM publications. These publications are listed in the Chapter 10, “Bibliography,” on page 591 with their full title and base order number. When they are referred to in text, a shortened version of the title is used.

- Statements, Clauses, Special Registers
- Program Structure
- Concepts
- What's New

Chapter 1. About ILE COBOL Language Reference

Read this section for information about the reference.

Read this section for information about the reference.

Who Should Use this Reference

This reference provides information about the ILE COBOL programming language on the IBM i (formerly OS/400) system. It is intended for people who have a basic understanding of data processing concepts and of the COBOL programming language.

Before using this reference, you should be familiar with certain IBM i system information:

- 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
- You should know how to operate your display station when it is attached to the IBM i system and running IBM i software. This means knowing about the IBM i 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 CL commands
  - Call utilities
  - Respond to messages

  To find out more about this operating system and its control language, refer to the *CL and APIs* section of the *Programming* category in the IBM i Information Center. You can also refer to the *CL Programming* publication.
- You should be familiar with the *Data Management* topic in the Information Center, which provides detailed descriptions of the entries and keywords needed to describe database files and certain device files external to the user's program.
- You should be familiar with the *DDS Reference* topic in the Information Center manual, which provides information on using data management support and 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 IBM i system:

  - The screen design aid (SDA) is used to design and code displays. This information is contained in *ADTS for AS/400: Screen Design Aid*.
  - The source entry utility (SEU), is a full-screen editor you can use to enter and update your source and procedure members. This information is contained in *ADTS for AS/400: Source Entry Utility*.
  - The programming development manager (PDM) utility is a list-processing tool you can use to work with lists of libraries, objects, members, and user-defined options. This information is contained in *ADTS/400: Programming Development Manager*.
- You should know how to use the application programming interfaces (APIs) provided with the IBM i operating system. This information is contained in the .
- You should know how to interpret displayed and printed messages. This information is contained in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.
- You should be familiar with the concepts and terminology of the Integrated Language Environment. This information is contained in the *ILE Concepts* manual.

Prerequisite and Related Information

Use the IBM i Information Center as your starting point for looking up IBM i technical information. You can access the Information Center in two ways:

- From the following Web site:

  ```
  http://www.ibm.com/systems/i/infocenter/
  ```

- From CD-ROMs that ship with your IBM i order.

The IBM i Information Center contains advisors and important topics such as CL commands, system application programming interfaces (APIs), logical partitions, clustering, Java™ , TCP/IP, Web serving, and secured networks. It also includes links to related IBM Redbooks and Internet links to other IBM Web sites such as the Technical Studio and the IBM home page.

The manuals that are most relevant to the ILE COBOL compiler feature are listed in the Chapter 10, “Bibliography,” on page 591.

How to Send Your Comments

Your feedback is important in helping to provide the most accurate and high-quality information. IBM welcomes any comments about this book or any other IBM i documentation.

- If you prefer to send comments by mail, use the the following address:

  IBM Canada Ltd. Laboratory
  Information Development
  8200 Warden Avenue
  Markham, Ontario, Canada L6G 1C7

  If you are mailing a readers' comment form from a country other than the United States, you can give the form to the local IBM branch office or IBM representative for postage-paid mailing.
- If you prefer to send comments by fax , use 1–845–491–7727, attention: RCF Coordinator.
- If you prefer to send comments electronically, use one of these e-mail addresses:
  - Comments on books:

    RCHCLERK@us.ibm.com
  - Comments on the IBM i Information Center:

    RCHINFOC@us.ibm.com

Be sure to include the following:

- The name of the book.
- The publication number of the book.
- The page number or topic to which your comment applies.

What's New

There have been several releases of ILE COBOL. The following is a list of enhancements made for each release since V3R1 up to the current release:

You can use this section to link to and learn about new ILE COBOL functions.

Note: The information for this product is up-to-date with the 7.2 release of ILE COBOL. If you are using a previous release of the compiler, you will need to determine what functions are supported on your system. For example, if you are using a 6.1 system, the functions new to the 7.2 release will not be supported.

What's New this Release?

The following list describes the enhancements made to ILE COBOL in 7.2:

- TIMESTAMP support of 0 to 12 fractional seconds

  A timestamp item can now have between 0 and 12 fractional seconds.

  - The following intrinsic functions will now allow PICOSECONDS as a duration when specified for a timestamp item: ADD-DURATION, EXTRACT-DATE-TIME, FIND-DURATION, and SUBTRACT-DURATION.
  - The SIZE keyword is allowed with FORMAT TIMESTAMP. The size can be 19 indicating zero fractional seconds or a value between 21 and 32 indicating between 1 and 12 fractional seconds.
- XML PARSE now has the capability to parse XML files that are greater than 16MB in size, provided that no individual document piece passed to the processing procedure is greater than 16MB. The following new XML-CODE values are associated with this change:
  - XML-CODE 62 indicates that the XML document exceeds 16,000,000 bytes.
  - XML-CODE 170 indicates that an XML event exceeds 16,000,000 bytes.
- PCML generation
  - PCML generation provides improved OCCURS DEPENDING ON array handling with the addition of a new "init" keyword that will be set to the maximum size of the array.
  - PCML generation provides automatic data-item naming in generated PCML for filler data items and unnamed items in a data structure, helping to enable web services to use generated PCML without first modifying it. The names for these data items will be *_filler_1*, *_filler_2*, and so on.

- National (Unicode) enhancements
  - Numeric national datatype is supported
  - A numeric literal can be specified on the VALUE clause for a numeric national data item
  - The figurative constant ZERO/ZEROS/ZEROES represents one or more national zero digits when used with national data items
  - National 'N' literals are supported when new PROCESS option NATIONALPICNLIT is specified
- The accuracy of numeric intrinsic functions NUMVAL and NUMVAL-C increases to 31 digits with compiler option ARITHMETIC(*EXTEND31) or PROCESS option EXTEND31.
- ARITHMETIC parameter for CRTBNDCBL / CRTCBLMOD:

  New *EXTEND31FULL option value provides the following features:
  - The accuracy of the following numeric intrinsic functions increases from floating-point accuracy of up to 15 digits to decimal floating-point accuracy of up to 34 digits: ANNUITY, MEAN, MEDIAN, MIDRANGE, NUMVAL, NUMVAL-C, PRESENT-VALUE, and VARIANCE.
  - The intermediate result of a fixed-point arithmetic expression can be up to 34 digits and numeric literals may have a maximum length of 34 digits.
- New PROCESS statement options:
  - NOCHGFLTRND / ALWCHGFLTRND

    Specifies whether or not COBOL will use the floating point rounding mode computational attribute specified by MI instruction SETCA. SETCA allows you to set the rounding mode of the result of a floating-point calculation to either round or truncate.
  - NATIONALPICNLIT

    Enables N" and N' as the opening delimiter for a national literal and enables elementary data items defined using the picture symbol N to have an implied USAGE NATIONAL clause.
  - EXTEND31FULL

Note: There may be screen captures in this guide that contain obsolete references to iSeries.

Changes to this Guide Since 7.1

This 7.2 guide, *IBM Rational Development Studio for i: ILE COBOL Language Reference*, SC09-2539-08, differs in several places from the 7.1 guide, SC09-2539-07. Most of the changes are related to the enhancements; others reflect minor technical corrections. To assist you in using this manual, technical changes and enhancements made in 7.2 are noted with a vertical bar (|).

What's New in 7.1?

The following list describes the enhancements made to ILE COBOL in 7.1:

- COMPUTATIONAL-5 (native binary) data type

  COMPUTATIONAL-5 or COMP-5 is a native binary data type now supported by the USAGE clause. COMP-5 data items are represented in storage as binary data, and can contain values up to the capacity of the native binary representation (2, 4, or 8 bytes). When numeric data is moved or stored into a COMP-5 item, truncation occurs at the binary field size rather than at the COBOL picture size limit. When a COMP-5 item is referenced, the full binary field size is used in the operation. This support will enhance portability to or from COBOL on other IBM platforms and operating systems.
- Ability to specify a non-numeric literal on the VALUE clause for a national data item.
- XML GENERATE performance improvements and PROCESS options

  Performance improvements have been made for XML GENERATE when the APPEND option is specified. Users who have a large number of data records to be appended into a data structure or into a stream file will benefit from these changes. The improvements include the addition of new PROCESS statement parameter XMLGEN with option values:

  - NOKEEPFILEOPEN / KEEPFILEOPEN

    Specify KEEPFILEOPEN to indicate that the XML stream file is to be left open and not closed when the XML GENERATE statement is complete, so that subsequent XML GENERATE FILE-STREAM APPEND statements can quickly append data to the stream file.
  - NOASSUMEVALIDCHARS / ASSUMEVALIDCHARS

    Specify ASSUMEVALIDCHARS to have XML GENERATE bypass the checking for special characters (less than "<", greater than ">", ampersand "&", and the single and double quote symbols), and for characters not supported by XML that would require being generated as hexadecimal. Otherwise normal checking will be done with the default NOASSUMEVALIDCHARS.
- Ability to encrypt the listing debug view

  A new CRTBNDCBL / CRTCBLMOD parameter is added to support the encryption of the listing debug view. DBGENCKEY specifies the encryption key to be used to encrypt program source that is embedded in debug views.
- Larger program support

  The CRTBNDCBL / CRTCBLMOD OPTIMIZE parameter now supports a new *NEVER option value. The *NEVER value allows larger programs to compile by not generating optimization code for the program. PROCESS statement option NEVEROPTIMIZE is also added.
- Support for the teraspace storage model

  The storage model for a program/module can now be specified using the new CRTBNDCBL / CRTCBLMOD parameter STGMDL with option values:
  - *SNGLVL specifies that the program/module is to be created with single-level storage model
  - *TERASPACE specifies that the program/module is to be created with teraspace storage model
  - *INHERIT specifies that the program/module is to inherit the storage model of its caller

  Additionally, the activation group parameter ACTGRP on the CRTBNDCBL command now has a new default option value:
  - *STGMDL: When STGMDL(*TERASPACE) is specified, the program will be activated into the QILETS activation group. For all other storage models, the program will be activated into the QILE activation group when it is called.
- New PROCESS statement options
  - ACTGRP is now available as a PROCESS statement parameter with option values:
    - STGMDL
    - NEW
    - CALLER
  - NEVEROPTIMIZE is now available as a PROCESS statement option
  - STGMDL is now available as a PROCESS statement parameter with option values:
    - INHERIT
    - SNGLVL
    - TERASPACE
  - XMLGEN is now available as a PROCESS statement parameter with option values:
    - NOKEEPFILEOPEN / KEEPFILEOPEN
    - NOASSUMEVALIDCHARS / ASSUMEVALIDCHARS

Note: There may be screen captures in this guide that contain obsolete references to iSeries.

What's New in V6R1?

The following list describes the enhancements made to ILE COBOL in V6R1:

- National UCS-2 CCSID support

  The NTLCCSID parameter has been added to the CRTCBLMOD and CRTBNDCBL commands, and to the PROCESS statement, to allow you to specify the UCS-2 CCSID to be used for National data items. With this parameter, you can specify a CCSID other than the default 13488, such as CCSID 1200, to be used for National items.
- PCML in module support
  - The PGMINFO parameter on the CRTCBLMOD and CRTBNDCBL commands has been enhanced to allow you to specify the location where you want to put the generated PCML. When the user specifies *PCML as the first parameter for the PGMINFO keyword, a second parameter specifying a location of *STMF, *MODULE, or *ALL, can also be specified. *STMF will cause the PCML to be put into the streamfile specified on the INFOSTMF parameter, *MODULE will cause the PCML to be put into the generated module, and *ALL will cause the PCML to be put in all of these locations.
  - PROCESS statement option PGMINFO

    This option allows the user to request that PCML be added to the module, and can be specified as PGMINFO(PCML MODULE). If the user had requested the PCML be added to a streamfile from the create command, the PCML will be added to both the module and the streamfile.
- Complex OCCURS DEPENDING ON (ODO) debugger support
  - Support has been added so the system debugger and the client debugger can now debug complex OCCURS DEPENDING ON arrays.
- Large Program Support
  - The compiler has been enhanced so that larger programs and programs containing a very large number of data items can now be compiled (subject to system limitations).

What's New in V5R4?

The following list describes the enhancements made to ILE COBOL in V5R4:

- XML support has been enhanced. A new statement, XML GENERATE, converts the content of COBOL data records to XML format. XML GENERATE creates XML documents encoded in Unicode UCS-2 or in one of several single-byte EBCDIC or ASCII CCSIDs. .
- Null-terminated nonnumeric literal

  Nonnumeric literals can be null-terminated. They can be used anywhere a nonnumeric literal can be specified except that null-terminated literals are not supported in "ALL literal" figurative constants.
- New CRTBNDCBL / CRTCBLMOD option

  *NOCOMPRESSDBG/*COMPRESSDBG specifies whether listing view compression should be performed by the compiler when DBGVIEW option *LIST or *ALL is specified.
- New intrinsic functions:
  - DISPLAY-OF
  - NATIONAL-OF
  - TRIM
  - TRIML
  - TRIMR

What's New in V5R3?

The following list describes the enhancements made to ILE COBOL in V5R3:

- Large VALUE clause support

  When the *NOSTDTRUNC compiler option is in effect, data items described with usage BINARY, or COMP-4 that do not have a picture symbol P in their PICTURE clause can have a value up to the capacity of the native binary representation.

- CONSTANT data type

  A CONSTANT data type is defined by specifying a level-01 entry containing the CONSTANT clause for a literal. The CONSTANT data item can then be used in place of the literal.

- XML support

  XML PARSE statement provides the interface to a high-speed XML parser that is part of the COBOL run time. The XML PARSE statement parses an XML document into its individual pieces and passes each piece, one at a time, to a user-written processing procedure.

  These XML special registers are used to communicate information between the XML parser and the user-written processing procedure:

  - XML-CODE
  - XML-EVENT
  - XML-NTEXT
  - XML-TEXT

- Alternate Record Key support

  The ALTERNATE RECORD KEY clause lets you define alternate record keys associated with indexed files. These alternate keys allow you to access the file using a different logical ordering of the file records.

- DBCS data item names (DBCS word support)
- 63 digit support
  - The maximum length of packed decimal, zoned decimal, and numeric-edited items has been extended from 31 to 63 digits.
  - The ARITHMETIC parameter on the CRTCBLMOD and CRTBNDCBL commands and on the PROCESS statement has a new EXTEND63 option.
- 7 new ANSI Intrinsic functions:
  - INTEGER
  - REM
  - ANNUITY
  - INTEGER-PART
  - MOD
  - FACTORIAL
  - RANDOM
- New CRTBNDCBL / CRTCBLMOD options:
  - *NOCRTARKIDX / *CRTARKIDX Specifies whether or not to create temporary alternate record key indexes if permanent ones cannot be found.
  - *STDINZHEX00 Specifies that data items without a value clause are initialized with hexadecimal zero.
  - *EXTEND63 option for the ARITHMETIC parameter increases the precision of intermediate results for fixed-point arithmetic up to 63 digits.
- New PROCESS statement options:
  - PROCESS statement option NOCOMPRESSDBG/COMPRESSDBG indicates whether listing view compression should be performed by the compiler when DBGVIEW option *LIST or *ALL is specified
  - NOCRTARKIDX/CRTARKIDX
  - STDINZHEX00
  - EXTEND63 option for the ARITHMETIC parameter
- Program Status Structure

The program status structure is a predefined structure that contains error information when the COBOL program receives an error. The PROGRAM STATUS clause is used to specify the error information that is received.

What's New in V5R2?

The following list describes the enhancements made to ILE COBOL in V5R2:

- Recursive program support

  An optional RECURSIVE clause has been added to provide support for recursive programs. These are COBOL programs that can be recursively re-entered.

- Local Storage Section support

  A new data section that defines storage allocated and freed on a per-invocation basis has been added. You can specify the Local-Storage Section in both recursive and non-recursive programs.

- Java interoperability

  Two new features have been added to enhance Java interoperability. These include:

  – UTF8String intrinsic function

    This function provides the ability to convert strings to UTF-8 format.

  – PCML support

    New parameters have been added to the CRTCBLMOD and CRTBNDCBL commands to give users the ability to tell the compiler to generate PCML source for their COBOL program. When the user specifies PGMINFO(*PCML) and the name of a streamfile on the INFOSTMF parameter, the compiler will generate PCML into the specifed streamfile. The generated PCML makes it easier for Java programs to call this COBOL program, with less Java code.

- Additional intrinsic functions

  Several new intrinsic functions have been added to this release. These include:

  – Max
  – Median
  – Midrange
  – Min
  – ORD-Max
  – ORD-Min
  – Present Value
  – Range
  – Standard Deviation
  – Sum
  – Variance

- IFS

  ILE Cobol source stored in IFS stream files can be compiled. The SRCSTMF and INCDIR parameters have been added to the CRTCBLMOD and CRTBNDCBL commands to give users the ability to tell the compiler to compile from source stored in IFS stream files.

What's New in V5R1?

The following list describes the enhancements made to ILE COBOL in V5R1:

- UCS-2 (Unicode) support

  National data, a new type of data item, has been added to provide support for the coded character set specified in ISO/IEC 10646-1 as UCS-2. The code set is the basic set defined in the Unicode standard.

  - UCS-2 character set

    This coded character set provides a unique code for each character appearing in the principal scripts in use around the world. Each character is represented by a 16-bit (2-byte) code.

  - National data

    This new type of data item specifies that the item contains data coded using the UCS-2 code set. An elementary data item whose description contains a USAGE NATIONAL clause, or an elementary data item subordinate to a group item whose description contains a USAGE NATIONAL clause, is a national data item.

  - NTLPADCHAR compiler option and PROCESS statement option

    This option allows you to specify three values: the SBCS padding character, DBCS padding character, and national padding character. The appropriate padding character is used when a value is moved into a national datatype item and does not fill the national datatype item completely.

  - ALL national literal

    Allows the word ALL wherever a national hexadecimal literal is allowed, so that for example you could move all UCS-2 blanks into a national data item.

  - PROCESS statement option NATIONAL

    When this option is specified, elementary data items defined using the picture symbol N will have an implied USAGE NATIONAL clause. A USAGE DISPLAY-1 clause will be implied for these items if the compiler option is not used.

  - National hexadecimal literals

    Literals containing national data values may be specified using the syntax:

    ```
    NX"hexadecimal-character-sequence..."
    ```

  - Figurative constants

    The figurative constant SPACE/SPACES represents one or more UCS-2 single byte space characters (U+0020) when used with national data items.

- JAVA interoperability support
  - QCBLLESRC.JNI file

    This file provides the same definitions and prototypes that are provided in the JNI.h file, but written in COBOL rather than C.

  - Data mapping between Java and COBOL datatypes
- Mainframe portability support
  - NOCOMPASBIN/COMPASBIN PROCESS statement option indicates whether USAGE COMPUTATIONAL or COMP has the same meaning as USAGE COMP-3 or USAGE COMP-4.
  - NOLSPTRALIGN/LSPTRALIGN PROCESS statement option indicates whether data items with USAGE POINTER or PROCEDURE-POINTER are aligned at multiples of 16 bytes relative to the beginning of the record in the linkage section.
  - Complex OCCURS DEPENDING ON (ODO) support

    The following constitute complex ODO:

    - Entries subordinate to the subject of an OCCURS or an ODO clause can contain ODO clauses (table with variable length elements).
    - A data item described by an ODO can be followed by a non-subordinate data item described with ODO clause (variably located table).
    - Entries containing an ODO clause can be followed by non-subordinate items (variably located fields). These non-subordinate items, however, cannot be the object of an ODO clause.
    - The location of any subordinate or non-subordinate item, following an item containing an ODO clause, is affected by the value of the ODO object.

- The INDEXED BY phrase can be specified for a table that has a subordinate item that contains an ODO clause.

- The LICOPT parameter has been added to the CRTCBLMOD and CRTBNDCBL commands to allow advanced users to specify Licensed Internal Code options.

What's New in V4R4?

The following list describes the enhancements made to ILE COBOL in V4R4:

- Thread Safety Support

  Support for calling ILE COBOL procedures from a threaded application, such as Domino® or Java. The THREAD parameter has been added to the PROCESS statement, to enable ILE COBOL modules for multithreaded environments. Access to the procedures in the module should be serialized.

- 31-digit support
  - The maximum length of packed decimal, zoned decimal, and numeric-edited items has been extended from 18 to 31 numeric digits.
  - The ARITHMETIC parameter has been added to the CRTCBLMOD and CRTBNDCBL commands, and to the PROCESS statement to allow the arithmetic mode to be set for numeric data. This allows you to specify the computational behavior of numeric data.

- Euro currency support
  - The ability to specify more than one currency sign in a COBOL program to support the dual currency system that will be in effect for three years starting in January 1999 among the participating countries.
  - The ability to represent multi-character currency signs, so that the international currency signs (e.g. USD, FRF, DEM, EUR) as well as single-character currency signs (e.g. "$") can be specified for COBOL numeric edited fields.
  - The OPTION parameter values *MONOPIC/*NOMONOPIC have been added to the CRTCBLMOD and CRTBNDCBL commands, and MONOPIC/NOMONOPIC have been added to the PROCESS statement. This allows you to choose between a moncased or a case sensitive currency symbol in a PICTURE character-string.

What's New in V4R2?

The following list describes the enhancements made to ILE COBOL in V4R2:

- User-defined data types

  A user-defined data type is defined by specifying a level-01 entry containing the TYPEDEF clause; all entries that are subordinate to the level-01 entry are considered part of the user-defined data type. A user-defined data type can be used to define new data items of level-01, -77, or -02 through -49, by specifying a TYPE clause for the new data item, that references the user-defined data type.

- Program profiling support

  The PRFDTA parameter has been added to both the CRTCBLMOD and CRTBNDCBL commands, and to the PROCESS statement, to allow a program to be profiled for optimization.

- Null-values support

  Null-values support (by way of the NULL-MAP and NULL-KEY-MAP keywords) has been added to the following statements and clauses to allow the manipulation of null values in database records:
  - ASSIGN clause
  - COPY-DDS statement
  - DELETE statement
  - READ statement
  - REWRITE statement

  - START statement
  - WRITE statement.
- Locale support

  IBM i Locale objects (*LOCALE) specify certain cultural elements such as a date format or time format. This cultural information can be associated with ILE COBOL date, time, and numeric-edited items. The following new characters, clauses, phrases and statements were added to support this:

  - The LOCALE clause of the SPECIAL-NAMES paragraph
    - Associates an IBM i locale object with a COBOL mnemonic-name
  - The LOCALE phrase of a date, time, or numeric-edited item
    - Allows you to specify a locale mnemonic-name, so that the data item is associated with an IBM i locale object
  - Along with specific locales defined in the LOCALE clause of the SPECIAL-NAMES paragraph, a current locale, and a default locale have been defined. The current locale can be changed with the new SET LOCALE statement (Format 8).
    - A locale object is made up of locale categories, each locale category can be changed with the SET LOCALE statement.
  - Locale categories have names such as LC_TIME and LC_MONETARY. These names include the underscore character. This character has been added to the COBOL character set.
    - The SUBSTITUTE phrase of the COPY DDS statement has been enhanced to allow the underscore character to be brought in.

  The following new intrinsic functions allow you to return culturally-specific dates and times as character strings:

  - LOCALE-DATE
  - LOCALE-TIME.
- Additions to Century support

  The following enhancements have been made to the ILE COBOL Century support:

  - A new class of data items, class date-time, has been added. Class date-time includes date, time, and timestamp categories. Date-time data items are declared with the new FORMAT clause of the Data Description Entry.
  - Using COPY-DDS and the following values for the CVTOPT compiler parameter, IBM i DDS data types date, time, and timestamp can be brought into COBOL programs as COBOL date, time, and timestamp items:
    - *DATE
    - *TIME
    - *TIMESTAMP.
  - Using the CVTOPT parameter value *CVTTODATE, packed, zoned, and character IBM i DDS data types with the DATFMT keyword can be brought into COBOL as date items.
  - The following new intrinsic functions allow you to do arithmetic on items of class date-time, convert items to class date-time, test to make sure a date-time item is valid, and extract part of a date-time item:
    - ADD-DURATION
    - CONVERT-DATE-TIME
    - EXTRACT-DATE-TIME
    - FIND-DURATION
    - SUBTRACT-DURATION
    - TEST-DATE-TIME.

What's New in V3R7?

The following list describes the enhancements made to ILE COBOL in V3R7:

- Century support

  The capability for users to work with a 4-digit year has been added in the following statements and functions:

  – ACCEPT statement with the YYYYDDD and YYYYMMDD phrases

  – The following intrinsic functions convert a 2-digit year to a 4-digit year:

    - DATE-TO-YYYYMMDD
    - DAY-TO-YYYYDDD
    - YEAR-TO-YYYY

  – The following intrinsic functions return a 4-digit year:

    - CURRENT-DATE
    - DAY-OF-INTEGER
    - DATE-OF-INTEGER
    - WHEN-COMPILED

- Floating-point support

  The *FLOAT value of the CVTOPT parameter on the CRTCBLMOD and CRTBNDCBL commands allows floating-point data items to be used in ILE COBOL programs. Also, the affected statements (such as ACCEPT, DISPLAY, MOVE, COMPUTE, ADD, SUBTRACT, MULTIPLY, and DIVIDE) support floating-point.

- Data area support

  New formats of the ACCEPT and DISPLAY statements have been added to provide the ability to retrieve and update the contents of IBM i data areas.

- Intrinsic Functions

  The following intrinsic functions have been added:

  ACOS
  ASIN
  ATAN
  CHAR
  COS
  CURRENT-DATE
  DATE-OF-INTEGER
  DAY-OF-INTEGER
  DATE-TO-YYYYMMDD
  DAY-TO-YYYYDDD
  INTEGER-OF-DATE
  INTEGER-OF-DAY
  LENGTH
  LOG
  LOG10
  LOWER-CASE
  MEAN
  NUMVAL
  NUMVAL-C
  ORD
  REVERSE
  SIN
  SQRT
  TAN
  UPPER-CASE
  WHEN-COMPILED
  YEAR-TO-YYYY

- Binding Directory parameter—BNDDIR

  The BNDDIR parameter has been added to the CRTBNDCBL command to allow the specification of the list of binding directories that are used in symbol resolution.

- Activation Group parameter—ACTGRP

  The ACTGRP parameter has been added to the CRTBNDCBL command to allow the specification of the activation group that a program is associated with when it is called.

- Library qualified program objects and data areas

  The LIBRARY phrase has been added to the following ILE COBOL statements to allow IBM i program objects and data areas to be qualified with a library name:

  – CALL
  – CANCEL
  – SET
  – ACCEPT
  – DISPLAY

- Performance collection data

  The ENBPFRCOL parameter has been added to the CRTCBLMOD and CRTBNDCBL commands, and to the PROCESS statement to allow performance measurement code to be generated in a module or program. The data collected can be used by the system performance tool to profile an application's performance.

- New ILE debugger support

  The ILE debugger now allows you to:

  – Debug most OPM programs
  – Set watch conditions, which are requests to set breakpoints when the value of a variable (or an expression that determines the address of a storage location) changes.

What's New in V3R6/V3R2?

The following list describes the enhancements made to ILE COBOL in V3R6 and V3R2:

- New EXIT PROGRAM phrase

  The AND CONTINUE RUN UNIT phrase has been added to the EXIT PROGRAM statement to allow exiting of a calling program without stopping the run unit.

- New SET statement pointer format

  A new format of the SET statement has been added that enables you to update pointer references.

- DBCS Data Support

  You can now process Double Byte Character Set (DBCS) data in ILE COBOL. The ILE COBOL compiler supports DBCS, in which each logical character is represented by two bytes. DBCS provides support for ideographic languages, such as the IBM Japanese Graphic Character Set, Kanji.

- Support for CALL...BY VALUE and CALL...RETURNING

  CALL...BY VALUE and CALL...RETURNING gives you the ability to pass arguments BY VALUE instead of BY REFERENCE and receive RETURN values. This allows for greater ease of migration, and improved interlanguage support as ILE C for IBM i and ILE RPG for IBM i both support CALL... BY VALUE and CALL...RETURNING.

- Support of the BY VALUE and RETURNING phrases of the PROCEDURE DIVISION Header

  The BY VALUE phrase of the PROCEDURE DIVISION header allows COBOL to receive BY VALUE arguments from a calling COBOL program or other ILE language such as RPG, C, or C++. The RETURNING phrase of the PROCEDURE DIVISION header allows COBOL to return a VALUE to the calling ILE procedure.

What's New in V3R1?

The following list describes the enhancements made to ILE COBOL in V3R1:

- EXTERNAL data items

  You can define data items that are available to every program in the ILE COBOL run unit by using the EXTERNAL clause. No longer do you need to pass all variables that are to be shared across programs as arguments on the CALL statement. This support encourages greater modularity of applications by allowing data to be shared without using arguments and parameters on the CALL statement.

- EXTERNAL files

  You can define files that are available to every program in the run unit. You can seamlessly make I/O requests to the same file from any ILE COBOL program within the run unit that declares the file as EXTERNAL. For external files there is only one file cursor regardless of the number of programs that use the file. You can share files across programs, and thereby develop smaller, more maintainable programs. Using EXTERNAL files provides advantages over using shared open files since only one OPEN and CLOSE operation is needed for all participating programs to use the file. However, an EXTERNAL file cannot be shared among different activation groups nor with programs written in other programming languages.

- Nested Source Programs

  An ILE COBOL source program can contain other ILE COBOL source programs. These contained programs may refer to some of the resources, such as data items and files, of the programs within which they are contained or define their own resources locally, which are only visible in the defining program. As the ILE COBOL programs are themselves resources, their scope is also controlled by the nesting structure and the scope attribute attached to the program. This provides greater flexibility in controlling the set of ILE COBOL programs that can be called by an ILE COBOL program. Nested ILE COBOL programs provides a mechanism to hide resources that would otherwise be visible.

- INITIAL Clause

  You have a mechanism whereby an ILE COBOL program and any programs contained within it are placed in their initial state every time they are called. This is accomplished by specifying INITIAL in the PROGRAM-ID paragraph. This provides additional flexibility in controlling the COBOL run unit.

- REPLACE statement

  The REPLACE statement is useful to replace source program text during the compilation process. It operates on the entire file or until another REPLACE statement is encountered, unlike the COPY directive with the REPLACING phrase. The REPLACE statements are processed after all COPY statements have been processed. This provides greater flexibility in changing the ILE COBOL text to be compiled.

- DISPLAY WITH NO ADVANCING statement

  By using the NO ADVANCING phrase on the DISPLAY statement, you have the capability to leave the cursor following the last character that is displayed. This allows you to string together items to be displayed on a single line from various points in the ILE COBOL program.

- ACCEPT FROM DAY-OF-WEEK statement

  ILE COBOL now allows you to accept the day of the week (Monday = 1, Tuesday = 2 ...) and assign it to an identifier. This support complements the existing ACCEPT FROM DAY/DATE/TIME support.

- SELECT OPTIONAL clause for Relative Files

  This allows for the automatic creation of relative files even when the file is opened I-O. This extends the support that is already available for sequential files.

- Support for Nested COPY statements

  Copy members can contain COPY statements thereby extending the power of the COPY statement. If a COPY member contains a COPY directive, neither the containing COPY directive nor the contained COPY directive can specify the REPLACING phrase.

- Enhancements to Extended ACCEPT and DISPLAY statements

  You can work with tables on the Extended ACCEPT statement. This allows you to easily and selectively update the elements of the table.

Variable length tables are also allowed on the Extended ACCEPT and DISPLAY statements.

Also, the SIZE clause is supported on the extended ACCEPT statement.

- Procedure-pointer support

  Procedure-pointer is a new data type that can contain the address of an ILE COBOL program or a non-ILE COBOL program. Procedure-pointers are defined by specifying the USAGE IS PROCEDURE-POINTER clause on a data item. This new data type is useful in calling programs and or ILE procedures that are expecting this type of data item as its parameter. Procedure-pointer data items can also be used as the target of a CALL statement to call another program.

- New Special Registers
  - RETURN-CODE special register

    Allows return information to be passed between ILE COBOL programs. Typically, this register is used to pass information about the success or failure of a called program.
  - SORT-RETURN special register

    Returns information about success of a SORT or MERGE statement. It also allows you to terminate processing of a SORT/MERGE from within an error declarative or an input-output procedure.

- New Compiler options
  - *PICGGRAPHIC/*NOPICGGRAPHIC

    *PICGGRAPHIC is a new parameter for the CVTOPT option which allows the user to bring DBCS data into their ILE COBOL program.
  - *IMBEDERR/*NOIMBEDERR option

    *IMBEDERR is a new compiler option which includes compile time errors at the point of occurrence in the compiler listing as well as at the end of the listing.
  - *FLOAT/*NOFLOAT

    *FLOAT is a new parameter for the CVTOPT option which allows you to bring floating-point data items into your ILE COBOL programs with their DDS names and a USAGE of COMP-1 (single-precision) or COMP-2 (double-precision).
  - *NOSTDTRUNC/*STDTRUNC option

    *NOSTDTRUNC is a new compiler option which suppresses the truncation of values in BINARY data items. This option is useful in migrating applications from IBM System/390® (S/390®).
  - *CHGPOSSGN/*NOCHGPOSSGN option

    This option is useful when sharing data between the IBM i and IBM S/390®. This option is provided for IBM System/390 compatibility. It changes the bit representation of signed packed and zoned data items when they are used in arithmetic statements or MOVE statements and the values in these data items are positive.

- Quoted system names support

  Support has been added to allow literals where system-names are allowed. You can use whatever names the system supports and is no longer limited to valid COBOL names.

- There is no COBOL limit on the following functions as these are now determined by system constraints.
  - Number of declared files.
  - Number of parameters on the CALL statement and on the Procedure Division USING phrase. A system limit of 400 for ILE procedures and 255 for program objects does apply here.
  - Number of SORT-MERGE input files and the number of SORT-MERGE keys. The maximum number of SORT-MERGE input files is 32 and the maximum length of the SORT-MERGE key is 2000 bytes.

- START with NO LOCK statement.

By using the NO LOCK phrase on the START statement, the file cursor will be positioned on the first record to be read without placing a lock on the record. This support is provided for indexed and relative files and complements the READ with NO LOCK function that is already available.

Note: START with NO LOCK is a new statement in both ILE COBOL and OPM COBOL/400.

- Static procedure call support

  You can develop your applications in smaller, better maintainable module objects, and link them together as one program object, without incurring the penalty of dynamic program call overhead. This facility, together with the common runtime environment provided by the system, also improves your ability to write mixed language applications. The ILE programming languages permits the binding of C, RPG, COBOL, and CL into a single program object regardless of the mix of source languages.

  New syntax on the CALL literal statement and a new compiler option have been added to ILE COBOL to differentiate between static procedure calls and dynamic program calls.

- Variable Length Record support (RECORD IS VARYING Clause)

  You can define and easily use different length records on the same file using standard ANSI COBOL syntax. Not only does this provide great savings in storage but it also eases the task of migrating complex applications from other systems.

- Expanded compiler limits

  ILE COBOL now offers expanded compiler limits:

  – size of group and elementary data items
  – size of fixed and variable length tables
  – number of nesting levels for conditional statements
  – number of operands in various Procedure Division statements

ILE COBOL Syntax Notation

ILE COBOL 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:

  ```
  MOVE
  ```

  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 ILE COBOL reserved words, see “Appendix E. ILE COBOL Reserved Word List” on page 555.
- 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:

  ►►—
  indicates the beginning of a statement

  —►
  indicates that the statement syntax is continued on the next line.

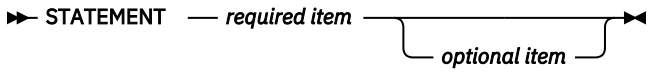
  ►—
  indicates that a statement is continued from the previous line.

  —►◄
  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.

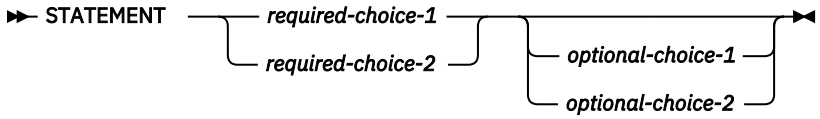
  Note: Statements within a diagram of an entire paragraph will not start with ►►— and end with —►◄ unless their beginning or ending coincides with that of the paragraph.

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

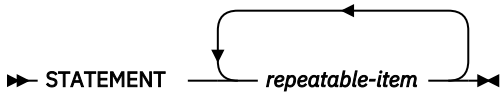


- When you can choose from two or more items, they appear vertically, in a stack.

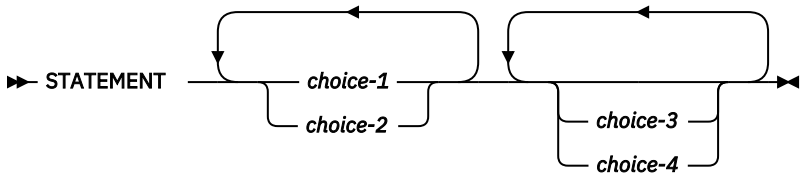
  If you must choose one of the items, one item of the stack appears on the main path. If choosing an item is optional, the entire stack appears below the main path:



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

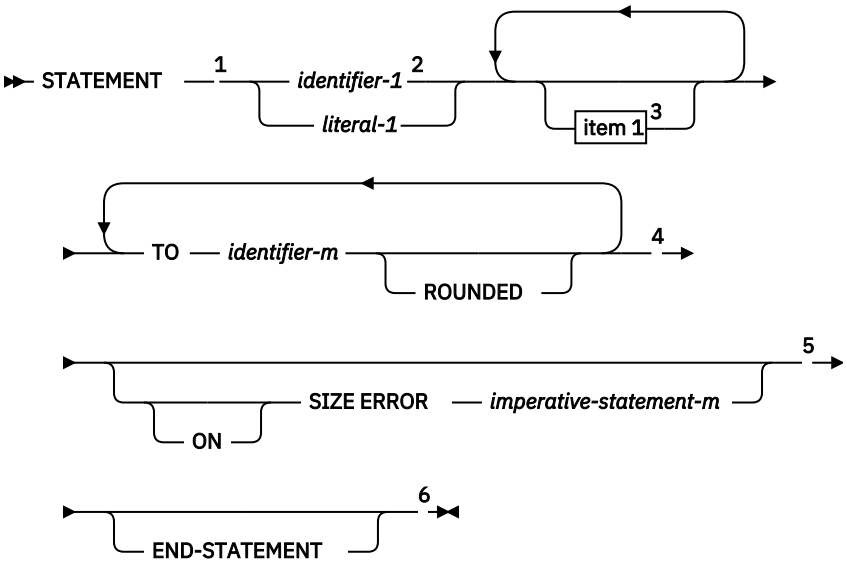


- A repeat arrow above a stack of required or optional choices indicates that you can make more than one choice from the stacked items, or repeat a single choice:

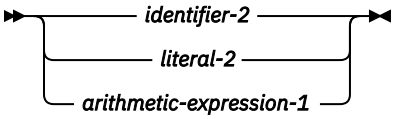


- An item that appears between two vertical bars identifies a fragment of the syntax that is defined in greater detail elsewhere in the diagram.

The following example shows how the syntax diagram conventions are used:



item 1



Format

Notes:

[1] The STATEMENT key word must be specified and coded as shown.

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

[3] The item 1 fragment is optional; it can be coded or not, as required by the application. If item 1 is coded, it can be repeated with each entry separated by one or more COBOL separators. Entry selections allowed for this fragment are described at the bottom of the diagram.

[4] The operand identifier-m and associated TO key word are required and can be repeated with one or more COBOL separators separating each entry. Each entry can be assigned the key word ROUNDED.

[5] The ON SIZE ERROR phrase with associated imperative-statement-m are optional. If the ON SIZE ERROR phrase is coded, the key word ON is optional.

[6] The END-STATEMENT key word can 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 ILE COBOL 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 ILE COBOL language extensions to ANSI X3.23b-1985 COBOL are indicated.

[IBM Extension] ILE COBOL language extensions to ANSI X3.23b-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 ILE COBOL compiler, are indicated in the syntax diagram with a footnote.

Obsolete Language Elements

Obsolete language elements are ILE COBOL language elements in the X3.23b-1993 COBOL standard that will be deleted from the next revision of this standard. Obsolete language elements are only syntax checked by the ILE COBOL compiler.

DBCS Notation

DBCS character strings in literals, comments are delimited by shift-out (represented by <) and shift-in (represented by >) characters. The EBCDIC codes for these characters are X'0E' and X'0F' respectively. Double-byte characters are represented thusly: D1D2D3, while DBCS literals look like this: G"<D1D2D3>". If you specify the APOST option, you may use single quotes instead.

Industry Standards

ILE COBOL is designed to support Standard COBOL. Standard COBOL refers to the COBOL programming language as defined in the document entitled American National Standard for Information Systems - Programming Language - COBOL, ANSI X3.23-1985, ISO 1989:1985, updated with the content of the following documents, in the order they are listed:

- ANSI X3.23a-1989, American National Standard for Information Systems - Programming Language - Intrinsic Function Module for COBOL and ISO 1989:1985/ Amd.1:1992
- Programming Languages - COBOL, AMENDMENT 1: Intrinsic function module
- ANSI X3.23b-1993, American National Standard for Information Systems - Programming Language - Correction Amendment for COBOL
- ISO/IEC 1989 DAM2 Programming Languages - COBOL, AMENDMENT 2: Correction and clarification amendment for COBOL.
- *FIPS Publication 21-4, Federal Information Processing Standard 21-4, COBOL*

From this point on, the term Standard COBOL will be used to refer to the ANSI standard just described.

Portions of this manual are copied from the *X3.23b-1993, American National Standard for Information Systems - Programming Language - COBOL* and *ISO 1989:1985/Amd 2:1994, Programming languages - COBOL* and are reproduced with permission from these publications (copyright 1985 by the American National Standards Institute), copies of which you can purchase from the American National Standard Institute at 1430 Broadway, New York, New York, 10018.

The COBOL language is maintained by the ANSI Technical Committee X3J4.

Refer to Appendix A of the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information on the industry standards supported by the ILE COBOL compiler.

An 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:

- Programming for the UNIVAC® I and II, Data Automation Systems copyrighted 1958, 1959, by Unisys 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.

Concepts

For help on the ILE COBOL language concepts, select from the following topics.

- Characters, character-strings, words, literals, comments, and separators
- Sections and paragraphs
- Reference format for a COBOL source line (Sequence Number Area, Indicator Area, Area A, Area B, and Comment Area)
- Methods of referencing data and procedures
- Types of COBOL names and their scope
- Methods of transferring control of program flow from one statement to another statement
- User-defined data types
- Supporting Information

Supporting Information

This section provides the following supporting information:

- ILE COBOL Compiler Limits
- Intermediate Results and Arithmetic Precision
- EBCDIC and ASCII Collating Sequences
- ILE COBOL Function-Name and Context-Sensitive Word List
- ILE COBOL Reserved Word List
- Attribute Data Formats
- ACCEPT/DISPLAY and COBOL/2 Considerations
- Bibliography

Chapter 2. COBOL Language Structure

Characters

In COBOL, the indivisible unit of data is the **character**. The letters of the alphabet, digits, and special characters that form the COBOL character set are shown in Table 1 on page 23.

The Integrated Language Environment* (ILE*) COBOL * language is restricted to the defined character set. The contents of nonnumeric literals, comment lines, comment entries, and the values held in data items, can include any of the characters from the character set currently specified for the system (by the primary source file's code character set identifier (CCSID)).

[IBM Extension] Characters from the Double-Byte Character Set (DBCS) are valid characters in certain COBOL character-strings. Double-byte characters occupy two adjacent bytes to represent one character. (See the DBCS information under "Character-Strings" on page 24 for more information.) [End of IBM Extension]

Individual characters are joined to form character strings, separators, and text words.

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 string of one or two contiguous punctuation characters used to delimit character-strings. Separators are described in detail under "Separators" on page 38.

A **text word** is a character or sequence of contiguous characters between margin A (between column 7 and column 8) and margin R (between column 72 and column 73) in a COBOL library, source program, or in pseudo-text that is any of the following:

- A separator, except for spaces, pseudo-text delimiters, and the delimiters of non-numeric literals
- A literal, including any required delimiters
- Any other sequence of contiguous COBOL characters bounded by separators, except comment lines and the word COPY.

Table 1. COBOL Characters—Their Meanings and Uses

Character	Meaning	Use
A–Z	Alphabet (uppercase)	Alphabetic characters
a–z	Alphabet (lowercase)	Alphabetic characters
0–9	Arabic numerals (digits)	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

Table 1. COBOL Characters—Their Meanings and Uses (continued)		
Character	**Meaning**	**Use**
*	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 Element of COBOL word
_	Underscore	Element of user-defined word
/	Stroke or Slash	Arithmetic operator Editing character Continuation character
,	Comma	Editing character Punctuation character
>	Greater than	Relation character
=	Equal sign	Punctuation character Relation character
"	Quotation mark	Punctuation character
'	Apostrophe	Punctuation character

Note:

1. The Apostrophe (') and underscore (_) characters are IBM extensions.
2. Certain other characters may be required within non-numeric literals used to define the names of system objects, or date and time formats:

   The characters # and @ are valid elements within IBM i system names.

   The characters @ and % are conversion specifiers that may be used when defining a date or time format.

Character-Strings

You can use character-strings containing single-byte characters to form:

- COBOL words with DBCS character
- COBOL words
- Literals
- PICTURE character-strings
- Comment-entry text

[IBM Extension]

You can use character-strings containing user-defined DBCS words to form:

- Literals
- Comment-entry text

DBCS character-strings are constructed using characters from the Double-Byte Character Set. DBCS character-strings can be embedded into nonnumeric strings, including mixed literals. Embedded DBCS character-strings must be delimited by an opening "shift-out" control character and a closing "shift-in" control character. DBCS literals may include characters that range from X'00' to X'FE' for both bytes. DBCS literals cannot include X'0F7F' (or X'0F7D' if the APOST option is selected).

[End of IBM Extension]

COBOL Words with DBCS Character

The following are the rules for forming user-defined words from DBCS characters:

Use of shift-out shift-in characters:
DBCS user-defined words begin with a shift-out character and end with a shift-in character.

Value range:
DBCS user-defined words can contain characters whose values range from hex 41 to hex FE for both bytes.

Contained characters:
DBCS user-defined words can contain only DBCS characters. DBCS user-defined words can contain characters that correspond to single-byte EBCDIC characters and those that do not correspond to single-byte EBCDIC characters. DBCS characters that correspond to single-byte EBCDIC characters follow the normal rules for COBOL user-defined words; that is, the characters A - Z, a - z, 0 - 9, and the hyphen (-) are allowed. The hyphen cannot appear as the first or last character. Any of the DBCS characters that have no corresponding single-byte EBCDIC character can be used in DBCS user-defined words.

Continuation rules:
DBCS words cannot be continued across lines.

Maximum length:
14 DBCS characters.

User-defined words:
The following types of user-defined words are supported in ILE COBOL. The second column indicates whether DBCS characters are allowed in words of a given type.

Type of user-defined word	DBCS characters allowed
Alphabet-name	Yes
Class-name (of data)	Yes
Condition-name	Yes
Data-name	Yes
File-name	Yes
Index-name	Yes
Library-name	No
Mnemonic-name	Yes
Paragraph-name	Yes
Program-name	No
Record-name	Yes
Section-name	Yes

Text-name	No

Note: In order to use the user-defined COBOL Words with DBCS character, PROCESS option GRAPHIC must be in effect, otherwise the DBCS words will be treated as invalid characters.

COBOL Words

COBOL words must be character-strings from the set of letters, digits, the hyphen, and the underscore. (The hyphen and the underscore cannot appear as the first or last character, however.) In the ILE COBOL language, each lowercase letter is generally equivalent to the corresponding uppercase letter.

The five types of COBOL words are:

- User-defined words
- System-names
- Function-names
- [IBM Extension] Context-sensitive words [End of IBM Extension]
- Reserved words

The following rules apply to all COBOL words that are not special character words within a source program:

- The maximum length of a COBOL word is 30 characters.
- With the exception of LENGTH, RANDOM, SUM, and WHEN-COMPILED, a reserved word cannot be used as a user-defined word, a system-name, a context-sensitive word, or a function-name.
- The same COBOL word however, may be used as two or more of the following types of ILE COBOL words:
  - User-defined word
  - System-name
  - Function-name
  - [IBM Extension] Context-sensitive word [End of IBM Extension]

  The classification of a specific occurrence of such a COBOL word is determined by the context of the phrase in which it occurs.

User-Defined Words

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 constant-name data-name file-name index-name locale-name mnemonic-name program-name (for contained programs and outermost programs that are called using the PROCEDURE linkage convention) record-name routine-name type-name	Each word must contain at least one letter.
library-name program-name (for outermost programs that are called using the PROGRAM linkage convention) text-name	Each word must contain at least one letter. The first 10 characters must form a unique word.
paragraph-name section-name	The word need **not** contain an alphabetic character.
Level-numbers: 01-49,66,77,88 Segment-numbers: 00-99	Each word must be a 1- or 2-digit integer; it does not have to be unique. (Segmentation information is syntax checked only.)

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

Constraints for Referencing User-Defined Words

In general, a user-defined word belongs to one, and only one, of the types listed in the preceding table. A user-defined word must also be unique within the type to which it belongs.

There are two exceptions to the general rule:

- A level-number or segment-number does not need to be unique. The user-defined word for any level-number or segment-number can be identical to the user-defined word for another level-number or segment number.
- A user-defined word can be duplicated within *one* of the following, provided that uniqueness of reference can be maintained:
  - The group comprising condition-names, data-names, and record-names
  - Paragraph-names
  - Text-names

  For more information about ensuring the uniqueness of reference for such names, see "Methods of Data Reference" on page 45.

The following types of user-defined words can be referenced by statements and entries in that program in which the user-defined word is declared:

- paragraph-name
- section-name

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

- library-name
- program-name
- text-name

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

- alphabet-name
- class-name
- condition-name
- mnemonic-name

System-Names

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

- computer-name
- language-name
- implementer-name (which includes environment-name and assignment-name)
- locale-name

Computer-name can be written in DBCS characters, but the other system-names cannot.

Function-Names

A **function-name** is a word that is one of a specified list of words used in COBOL source programs.

A function-name specifies the mechanism provided by ILE COBOL to determine the value of an intrinsic function.

With the exception of the words LENGTH, RANDOM, SUM, and WHEN-COMPILED, a word that is a function-name, in a different context, can appear in a program as a user-defined word, a system-name, or a context-sensitive word.

[IBM Extension] Context-Sensitive Words

A **context-sensitive word** is a COBOL word that is formed according to rules for reserved words, and may be used as specified in the general formats. The same word may also be used as a function-name, a user-defined word, or a system-name.

ILE COBOL context-sensitive words are listed in “Appendix D. ILE COBOL Function-Name and Context-Sensitive Word List” on page 552.

[End of IBM Extension]

Reserved Words

A **reserved word** is a character-string with a predefined meaning in a COBOL source program, and can be used only as specified in the language defined formats.

ILE COBOL reserved words are listed in "Appendix E. ILE COBOL Reserved Word List" on page 555.

There are five types of reserved words:

- Keywords
- Optional words
- Special character words
- Figurative constants
- Special registers

Keywords

Keywords are reserved words that are required within a given clause, entry, or statement.

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 meaning or execution of the program. Optional words are shown in formats as uppercase, but appear below the main path.

Special Character Words

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

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

  See "Arithmetic Operators" on page 224.
- **Relational operators**: < > = <= >=

  See "Relation Condition" on page 228.

Figurative Constants

Figurative constants are reserved words that name and refer to specific constant values.The reserved words for figurative constants and their meanings are:

ZERO/ZEROS/ZEROES
Represents one of the following, depending on the context:

- The numeric value zero (0)
- One or more occurrences of the nonnumeric character zero (0)
- [IBM Extension] The Boolean value B"0" [End of IBM Extension]

SPACE/SPACES
Represents one or more blanks or spaces;treated as a nonnumeric literal. SPACES represent one or more double-byte spaces when used with DBCS data items. SPACES represent one or more single-byte UCS-2 spaces when used with national data items.

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 NATIVE and EBCDIC collating sequences, the character is X'FF'; for the STANDARD-1 and STANDARD-2 collating sequences, the character is X'07'; 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 NATIVE, EBCDIC, STANDARD-1, and STANDARD-2 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.

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] The literal used in an ALL literal can be a Boolean literal, DBCS literal, or national literal. [End of IBM Extension]

[IBM Extension]

NULL/NULLS
Represents a value used to indicate that a data item defined with the USAGE IS POINTER clause, USAGE IS PROCEDURE-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 ILE COBOL language, a value of NULL is undefined.

[End of IBM Extension]

The singular and plural forms of ZERO, SPACE, HIGH-VALUE, LOW-VALUE, QUOTE, and NULL 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 can be used wherever ‘literal’ appears in a format, except where explicitly prohibited. When a numeric literal appears in a format, only the figurative constant ZERO can be used. Figurative constants are not allowed as function arguments except in an arithmetic expression, where they are arguments to a function.

[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 one (1) or to the number of character positions in the associated data item, whichever is greater.
- 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 specific COBOL features. Each such storage area has a fixed name, and must not be further defined within the program.

In the general formats of this specification, a special register can be used, unless otherwise restricted, wherever a data-name or identifier is specified provided that the special register is the same category as the data-name or identifier. If qualification is allowed, special registers can be qualified as necessary to provide uniqueness.

When control of a program is transferred for the first time from one program to another within the run unit by the CALL statement, the compiler initializes the special register fields to their initial values. The RETURN-CODE and SORT-RETURN special registers are reset to their initial values in the following instances:

- Whenever the CANCEL statement is invoked to initialize a referenced subprogram
- For programs that possess the INITIAL attribute
- For programs that possess the RECURSIVE attribute

In all other cases, the special registers are not reset to their initial values. Instead, they remain unchanged from the value retained the previous time program control was transferred via the CALL statement.

You can specify an alphanumeric register in a function wherever an alphanumeric argument is allowed, unless specifically prohibited.

You can specify a numeric special register in a function wherever a numeric argument is allowed, unless specifically prohibited.

Each special register is discussed in the section beginning on the indicated page.

Special Register
Page

DEBUG-ITEM
This register is syntax checked only.

LINAGE-COUNTER
"LINAGE-COUNTER Special Register" on page 148

[IBM Extension]

ADDRESS OF
"ADDRESS OF Special Register" on page 126

DB-FORMAT-NAME
"DB-FORMAT-NAME Special Register" on page 252

LENGTH OF
"LENGTH OF Special Register" on page 286

LOCALE OF
"LOCALE OF Special Register" on page 161

FORMAT OF
"FORMAT OF Special Register" on page 162

RETURN-CODE
"RETURN-CODE Special Register" on page 415

SORT-RETURN
"SORT-RETURN Special Register" on page 338

WHEN-COMPILED
"WHEN-COMPILED Special Register" on page 344

XML-CODE
"XML-CODE Special Register" on page 455

XML-EVENT
"XML-EVENT Special Register" on page 455

XML-NTEXT
"XML-NTEXT Special Register" on page 457

XML-TEXT
"XML-TEXT Special Register" on page 458

[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 page "Figurative Constants" on page 29). There are five types of literals:

[IBM Extension]

- Boolean
- DBCS
- National

[End of IBM Extension]

- Nonnumeric
- 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.

[End of IBM Extension]

[IBM Extension] DBCS Literals

DBCS literals have the following format:

Format

Format

G" or N"
The opening delimiter for a DBCS literal.

"
The closing delimiter for a DBCS literal.

When the NATIONALPICNLIT PROCESS option is in effect, the opening delimiter N" or N' identifies a national literal, and the rules specified in "Basic National Literals" on page 34 apply.

In general, the rules for forming a nonnumeric literal also apply to DBCS literals. The maximum length of DBCS literals, however, is 28 double-byte characters, and they cannot be continued across lines.

DBCS literals can be specified in the Data Division:

- In the VALUE clause of DBCS data description entries. If you specify a DBCS literal in a VALUE clause for a data item, the length of the literal must not exceed the size indicated by the data item's PICTURE clause. Explicitly defining a DBCS data item as USAGE DISPLAY-1 specifies that the data item is to be stored in character form, one character to each 2 bytes.
- With the JUSTIFIED clause.

DBCS literals can be specified in the Procedure Division:

- As the sending item when a DBCS or group item is the receiving item.
- In a relation condition when the comparand is a DBCS or group item.
- As the figurative constants SPACE/SPACES, ALL SPACE/SPACES, or ALL followed by a DBCS literal. These are the only figurative constants that can be DBCS literals.
- As an argument to an intrinsic function that supports DBCS.

DBCS literals can be specified wherever nonnumeric literals are allowed, **except** as a literal in the following:

- Identification Division
  - PROGRAM-ID paragraph
- Environment Division
  - ALPHABET clause
  - ASSIGN clause
  - CLASS clause
  - CURRENCY SIGN clause
  - LINKAGE clause
  - PADDING CHARACTER clause
  - RERUN clause
- Procedure Division
  - CALL statement (program-name)
  - CANCEL statement
  - END PROGRAM header
  - STOP statement
  - DROP statement
  - ACQUIRE statement
- COPY
  - COPY statement (text-name)
  - COPY statement (library-name).

[End of IBM Extension]

[IBM Extension] National Literals

National literals can be specified in the following places:

- In the VALUE clause of national data description entries
- As the sending item in the procedure division. For more information, see “MOVE Statement” on page 338.
- As an operand in the relation condition
- As an argument to the intrinsic functions than support national data.

ILE COBOL provides two national literal formats:

- Basic national literals
- National hexadecimal literals

The figurative constant SPACE/SPACES, ALL SPACE/SPACES can be a national literal. SPACE is single byte UCS-2 space (`NX"0020"`).

[End of IBM Extension]

[IBM Extension] Basic National Literals

When the NATIONALPICNLIT PROCESS option is in effect, the opening delimiter N" or N' identifies a national literal. A national literal is of the class and category national.

When the NATIONALPICNLIT PROCESS option is not in effect, the opening delimiter N" or N' identifies a DBCS literal, and the rules specified in "DBCS Literals" on page 32 apply.

Basic national literals have the following format:

Format

N"
The opening delimiter for a national literal.

character-data
The source text representation of the content of the national literal. *character-data* can contain any allowable character from the EBCDIC character set.

"
The closing delimiter for a national literal.

The enclosing 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 ("").

If the *APOST compiler option is in effect, the national literal must be enclosed by apostrophes (').

To include the quotation mark or apostrophe used in the opening delimiter in the content of the literal, specify a pair of quotation marks or apostrophes, respectively. For example

```
N'This literal''s content includes an apostrophe'
N'This literal includes ", which is not used in the opening delimiter'
N"This literal includes "", which is used in the opening delimiter"
```

The maximum length of a basic national literal is 256 characters, excluding the opening and closing delimiters. The literal must contain at least one character.

The source text representation of character-data is automatically converted to the national CCSID in effect for use at run time, for example, when the literal is moved to or compared with a data item of category national.

ALL followed by a basic national literal is a basic national literal.

[End of IBM Extension]

[IBM Extension] National Hexadecimal Literals

National hexadecimal literals have the following format:

Format

NX"
The opening delimiter for a national hexadecimal literal.

"
The closing delimiter for a national hexadecimal literal.

The hexadecimal character sequence consists of groups of four hexadecimal digits that map to a Universal Character Set Version 2 (UCS-2) or Unicode character.

The maximum length of national hexadecimal literals is 512 national characters.

ALL followed by a national hexadecimal literal is a national hexadecimal literal.

[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 256 characters.

A nonnumeric literal must be enclosed in quotation marks (").

If the *APOST compiler option is in effect, the nonnumeric literal must be enclosed by apostrophes (').

The enclosing 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 ("").

For example,

```
"THIS ISN""T WRONG"
```

[IBM Extension]

In an apostrophe literal, a double apostrophe ('') is reduced to a single apostrophe when the double apostrophe is also a delimiter.

For example,

```
'THIS ISN''T WRONG'
```

represents

```
THIS ISN'T WRONG
```

[End of IBM Extension]

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 130.)

[IBM Extension] Hexadecimal Literals

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 are characters that range from 0 to 9, a to f, 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 512 hexadecimal digits.

The continuation rules are the same as those for nonnumeric literals.

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

[End of IBM Extension]

[IBM Extension] Mixed Literals

Mixed literals are nonnumeric literals that combine single-byte and double-byte characters. Each string of double-byte characters must be delimited by an opening "shift-out" control character (hexadecimal 0E)

and a closing "shift-in" control character (hexadecimal 0F), to distinguish it from single-byte data. The control characters are included in the length of the mixed literal. A double-byte character string may consist solely of the two control characters.

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 character strings that are not valid mixtures of single-byte and double-byte characters. It is the user's responsibility to be certain that the statements are used correctly.

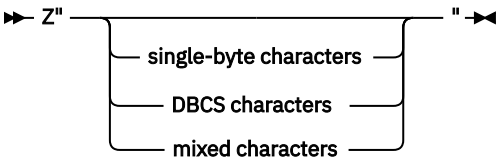
A mixed literal will only be recognized as such if the program is compiled using the GRAPHIC option of the PROCESS statement; otherwise, it will be treated as a simple non-numeric literal.

[End of IBM Extension]

[IBM Extension] Null-terminated nonnumeric literals

Nonnumeric literals can be null-terminated, with the following format:

Format



Z"

The opening delimiter for null-terminated notation of a nonnumeric literal. (If the compiler option *APOST or the PROCESS statement option APOST is specified, the opening delimiter is Z'.) Both characters of the opening delimiter for null-terminated literals (Z" or Z') must be on the same source line.

"

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

If a quotation mark is used in the opening delimiter, it must be used as the closing delimiter. Similarly, if an apostrophe is used in the opening delimiter, it must be used as the closing delimiter.

The content of the literal can include single-byte and/or double-byte characters, except that you cannot specify the single-byte character with the value X'00'. X'00' is the null character automatically appended to the end of the literal. The content of the literal is otherwise subject to the same rules and restrictions as a mixed literal.

The length of the string of single-byte and/or double-byte characters in the literal content can be 0 to 255 bytes. The actual length of the literal includes the terminating null character, giving a maximum length of 256 bytes.

A null-terminated nonnumeric literal has data class and category alphanumeric. It can be used anywhere a nonnumeric literal can be specified except that null-terminated literals are not supported in ALL literal figurative constants.

Avoid using a null-terminated literal to specify the external or internal object's name (such as program name, locale name, library name, procedure name, etc.) in a COBOL program; otherwise, compiler will replace the terminating null character by character "0", and a severity 20 error message will be issued to inform the user of this replacement.

The LENGTH intrinsic function, when applied to a null-terminated literal, returns the number of bytes in the literal prior to but not including the terminating null. (The LENGTH special register does not support literal operands.)

[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 that contains no decimal point. In some contexts, this literal is not permitted to have a negative value, or is not permitted to be zero. These restrictions, and any others that might be applicable, are included with the description of the format). The following rules apply:

- One through 18 digits are allowed when the (default) compiler option *NOEXTEND or the PROCESS statement option NOEXTEND is specified.
- [IBM Extension] One through 31 digits are allowed when the arithmetic mode compiler option *EXTEND31 or PROCESS statement option EXTEND31 is specified. [End of IBM Extension]
- [IBM Extension] One through 34 digits are allowed when the arithmetic mode compiler option *EXTEND31FULL or PROCESS statement option EXTEND31FULL is specified. [End of IBM Extension]
- [IBM Extension] One through 63 digits are allowed when the arithmetic mode compiler option *EXTEND63 or PROCESS statement option EXTEND63 is specified. [End of IBM Extension]
- Only one sign character is allowed. If included, it must be the left-most 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 right-most 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.

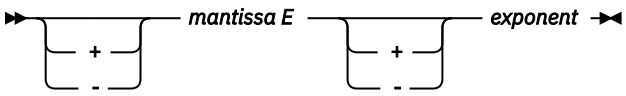
Every numeric literal is in the numeric data category. (Data categories are described under "Classes and Categories of Data" on page 130.)

[IBM Extension] Floating-Point Literals

Numeric literals may be either fixed-point or floating-point numbers. The rules for **floating-point literal values** are:

- A floating-point literal is written in the form:

  Format



- The sign is optional before the mantissa and the exponent; if you omit the sign, the compiler assumes a positive number.
- The mantissa can contain between 1 and 16 digits. A decimal point must be included in the mantissa.
- The exponent is represented by an E followed by an optional sign and one, two, or three digits.
- The magnitude of a floating-point literal value must fall between 2.225073858507201E-308 and 1.797693134862315E+308. For values outside of this range, an E-level diagnostic will be produced and the value will be replaced by either 0 or 1.797693134862315E+308, respectively.

  Note: The range for MVS™ COBOL is 0.54E-78 to 0.72E+76, and the range for OS/2® and AIX® is 2.225E-308 to 1.798E+308.

A floating-point literal is of class numeric and category internal floating-point. In general, a floating-point literal can be used wherever a numeric decimal literal is allowed.

[End of IBM Extension]

PICTURE Character-Strings

A **PICTURE character-string** consists of 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 13 on page 176.)

Comment-Entry Text

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
This form is described under "Optional Paragraphs" on page 72.

Comment line
This form is described on page "Comment Lines" on page 44.

Separators

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

The following is a list of the COBOL separator characters and their meaning.

b (space)
Space

,
Comma

.
Period

;
Semicolon

(
Left parenthesis

)
Right parenthesis

"
Quotation mark

==
Pseudo-text delimiter

:
Colon

[IBM Extension]

'
Apostrophe

B"
Opening delimiter for Boolean literal

X"
Opening delimiter for hexadecimal nonnumeric literal

G"
Opening delimiter for DBCS literal

N"
Opening delimiter for

- national literal if the NATIONALPICNLIT PROCESS option is in effect

- DBCS literal otherwise

NX"
Opening delimiter for national hexadecimal literal

Z"
Opening delimiter for null-terminated 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:

- The opening pseudo-text delimiter (where the preceding space is required).
- 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 [.␣], Comma [,␣], Semicolon [;␣]
A separator period, comma, or semicolon is composed of a period, comma, or semicolon followed by a space. 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 [( ] . . . [ )]
Except in pseudo-text, they must appear as balanced pairs of left and right parentheses. They delimit subscripts, a list of function arguments, reference modification, arithmetic expressions, and conditions.

Quotation marks [␣"] . . . ["␣]
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 43).

[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 [␣==]... literal-2 [==␣]
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 508 and “REPLACING Phrase” on page 509.)

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.

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

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

N" is a separator when used to describe a DBCS literal, or a national literal when NATIONALPICNLIT PROCESS option is in effect. The N must immediately precede the quotation mark.

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

Z" is a separator when used to describe a null-terminated nonnumeric literal. The Z 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. 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.

Clauses

A **clause** is an ordered set of consecutive COBOL character-strings that specifies an attribute of an entry.

Sentences

A **sentence** is a sequence of one or more statements, ending with a separator period.

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. For descriptions of the different types of statements, see:

- "Imperative Statements" on page 241
- "Conditional Statements" on page 243
- "Delimited Scope Statements" on page 243
- Chapter 8, "Compiler-Directing Statements," on page 507.

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 1 on page 41 shows the reference format for a COBOL 80-character source line.

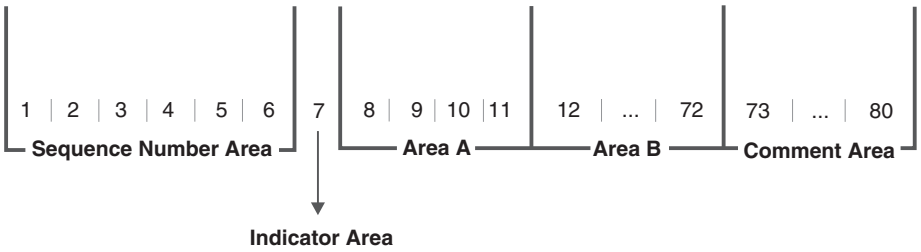


Figure 1. Reference Format for COBOL Source Line

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, and may consist of any character in the character set of the computer. Sequence numbers may be in any order, and they need not be unique.

[IBM Extension]

You can use sequence checking at compilation 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 lines from the previous line onto the current line (see the information on continuation lines on page "Continuation Lines" on page 43)
- The treatment of text as documentation or comments (see the information on comments on page "Comment Lines" on page 44)
- Debugging lines (see the information on debugging lines on page "Debugging Lines" on page 44)

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 (FD and SD) or level-number (01 and 77)
- DECLARATIVES and END DECLARATIVES
- END PROGRAM header

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

A **section header** indicates either the beginning of a series of paragraphs (in the Environment and Procedure Divisions), or the beginning of an entry (in the Data Division). For example, FILE-CONTROL in the former case, and FILE SECTION in the latter.

A section header must be immediately followed by a period except when Procedure Division segment numbers are specified. (Segmentation information is syntax checked only.)

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. 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 140.) 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 156.

DECLARATIVES and END DECLARATIVES

DECLARATIVES and END DECLARATIVES are keywords that begin and end the declaratives part of the source program. In the Procedure Division, each of these words must begin in Area A and be followed immediately by a separator period; no other text may appear on the same line. After END DECLARATIVES, no text may appear before the following section header. (See "Declaratives" on page 221.)

END PROGRAM Header

The **END PROGRAM** header, followed by program-name and a separator period, indicates the end of a COBOL program. Program-name must be identical to that of the corresponding PROGRAM-ID paragraph. Every COBOL program (except an outermost program that contains no nested programs and is not followed by another COBOL program in a sequence of COBOL programs) must end with this header.

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, and the amount is limited to the width of Area B. See also "Sections and Paragraphs" on page 40.

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, though the indicator area must contain a hyphen. If there is no hyphen 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 of a continued line is a single quotation mark in column 72, the first two nonblank characters in the continuation line must be two quotes to denote a single quote as part 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:

- Comment lines
- Debugging lines
- Blank lines
- Pseudo-text
- Compiler-directing statements other than the USE statement

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.

[IBM Extension] DBCS characters may be imbedded in a comment line, but cannot be continued to a following line. [End of IBM Extension]

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 Clause” on page 76.

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.

Compiler-Directing Statements

The following **compiler-directing** statements may start in Area A or Area B:

- [IBM Extension] *CONTROL(*CBL) [End of IBM Extension]
- COPY
- [IBM Extension] EJECT [End of IBM Extension]
- PROCESS
- REPLACE

[IBM Extension]

- SKIP1/2/3
- TITLE

[End of IBM Extension]

Comment Area (Columns 73 through 80)

The comment area is available for your own use; for example, to identify your program.

Data Reference and Name Scoping

The general concepts of Data Reference and Scoping of Names are important for the efficient and correct use of COBOL syntax. In particular, the scoping of names is important in using nested COBOL programs.

The first part of this section concentrates on the five methods of data reference:

- Qualification
- Subscripting
- Reference modification
- Function-identifier
- User-defined data types

The rest of this section concentrates on the scoping of names.

Methods of Data Reference

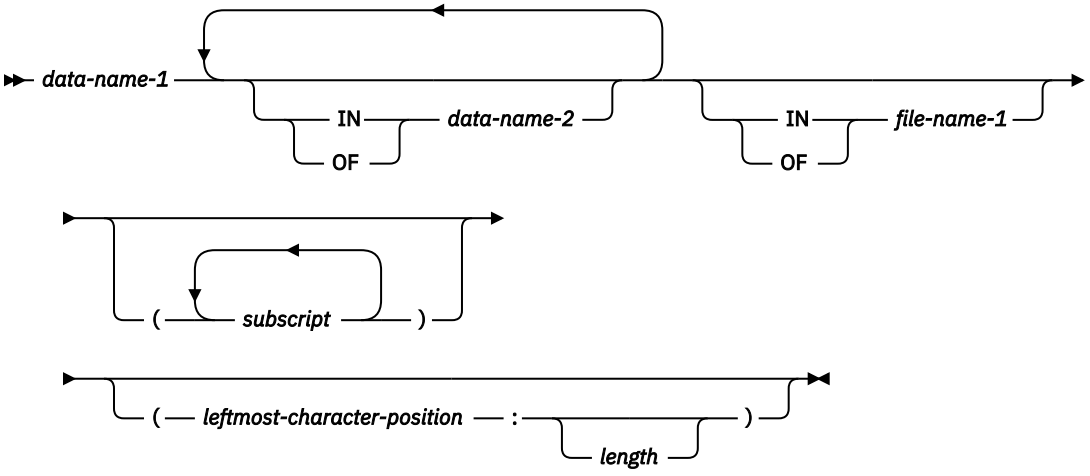
References to data and procedures can be either explicit or implicit.

Every user-defined name in a COBOL program names 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. Before looking at this, however, you need to understand the term **identifier**.

Identifier

In the syntax diagrams, the term **identifier** refers to a user-defined name that, if not unique in a program, must be followed by a syntactically correct combination of qualifiers, subscripts, or reference modifiers necessary for uniqueness of reference.

Format 1 - Identifier



data-name-1, data-name-2
Can be a record-name.

file-name-1
Must be identified by an FD or SD entry in the Data Division.

File-name-1 must be uniquely identifiable.

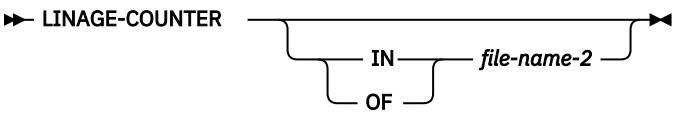
The following rules apply:

- Duplication of data-names must not occur in those places where the data-name cannot be made unique by qualification.
- In the same program, where data description entries for any two 01 level-number items have the same data-name, the external clause cannot be applied to either entry.
- In the same Data Division, the data description entries for any two data items for which the same data-name is specified must not include the GLOBAL clause.

There are two special cases for the identifier: **LINAGE-COUNTER** and **condition-name**.

LINAGE-COUNTER

Format 2 - LINAGE-COUNTER



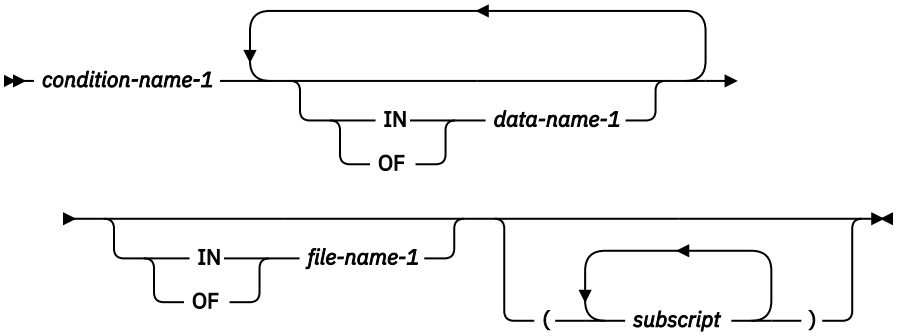
LINAGE-COUNTER
Must be qualified each time it is referenced if more than one file description entry containing a LINAGE clause has been specified in the source program.

file-name-2
Must be identified by an FD entry in the Data Division.

File-name-2 must be unique within this program.

condition-name

Format 3 - condition-name



condition-name-1
Can be defined in the Data Division or in the SPECIAL-NAMES paragraph within the Configuration Section of the Environment Division. If condition-name is defined in the Configuration Section, it can be referred to in the program containing the Configuration Section or in a nested program. If the condition-name is defined in the Data Division, it can be referenced according to the scoping rules for global and local names (see “Global and Local Names” on page 58).

If explicitly referenced, a condition-name must be unique or be made unique through qualification and/or subscripting except when the scope of names conventions by themselves ensure uniqueness of reference.

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

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

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

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

data-name-1
Can be a record-name.

file-name-1
Must be identified by an FD or SD entry in the Data Division.

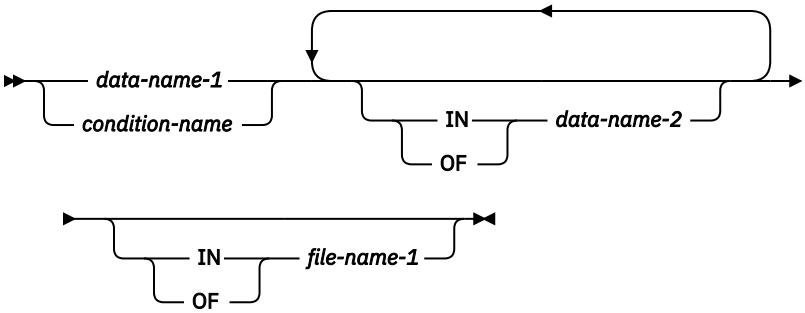
File-name-1 must be unique within this program.

Qualification

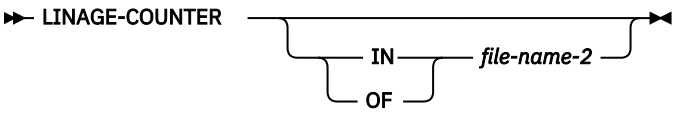
A name can be made unique if it exists within a hierarchy of names, and the name can be identified by specifying one or more higher-level names in the hierarchy. The higher-level names are called **qualifiers**, and the process by which such names are made unique is called **qualification**.

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.)

References to Data Division Names - Format 1



References to Data Division Names - Format 2



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

You 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

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.

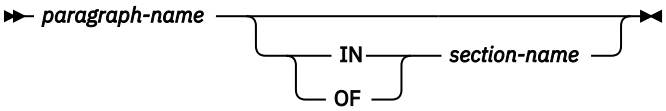
A data-name associated with a level-number 01, or with an FD or SD level indicator in the File Section, is the highest level in a data hierarchy. If referenced, it must be uniquely defined, because it cannot be qualified. Data items with level-numbers 02 through 49 are successively lower levels in a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

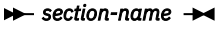
If explicitly referenced, a paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be

qualified when referred to within the section in which it appears. A paragraph-name or section-name appearing in a program cannot be referenced from any other program. A section-name, described in "Section" on page 222, is the highest (and only) qualifier available for a paragraph-name and must be unique.

References to Procedure Division Names - Format 1

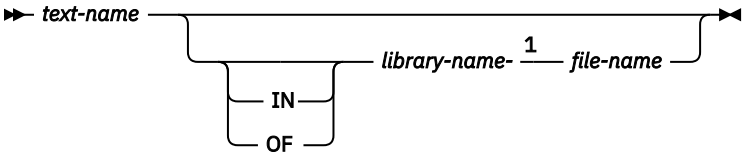


References to Procedure Division Names - Format 2



References to COPY Libraries

If more than one COBOL library is available to the compiler during compilation, text-name can be qualified each time it is referenced.



References to COPY Libraries - Format 3

Notes:

[1] Required hyphen between library-name-file-name to qualify

For rules on referencing COPY libraries, see "COPY Statement" on page 508.

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 189).

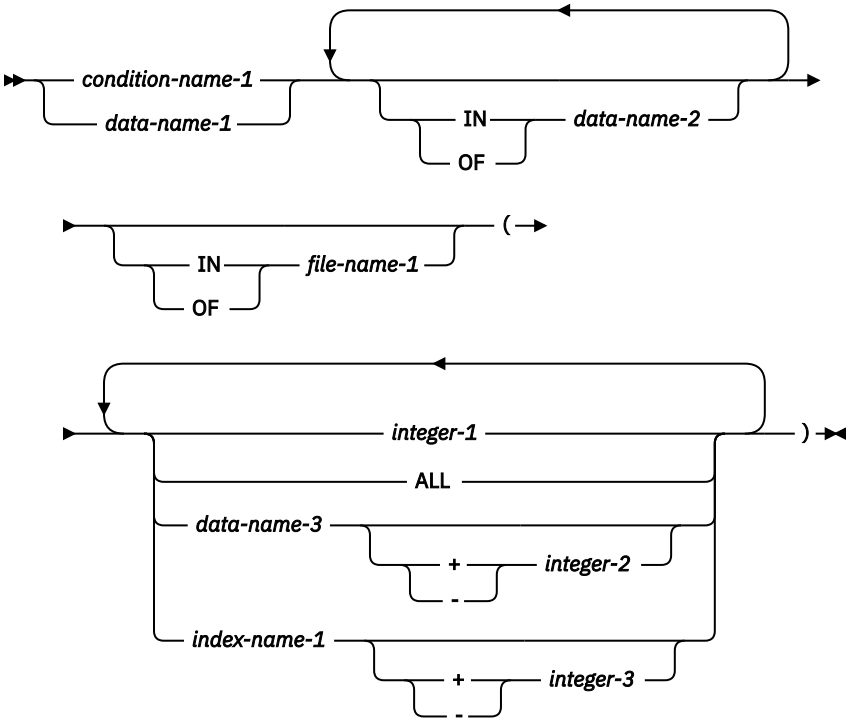
- If reference can be made unique by qualification, data-names may be defined in more than one place in a given program or compilation unit.
- If there is more than one combination of qualifiers that ensures uniqueness, then any of these combinations can be used.
- If referenced in the program, a section-name must be unique.
- If referenced in the program, a paragraph-name must be unique 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) can be qualified by the library-name and file-name in which it resides. (A hyphen is required between library-name and file-name, without any intervening spaces.) If no library is specified, the library list is searched. If no file-name is specified, QCBLLESRC is used.

  [IBM Extension] File-name is optional for the COPY statement, Format 1. If file-name is not specified, the default is QCBLLESRC. [End of IBM Extension]

Subscripting

Subscripting is a method of providing table references through the use of positive integers. A **subscript** is a positive integer (or integer data item) whose value specifies the occurrence number of a table element.

Subscripting - Format



condition-name-1
Must be subordinate to a data description entry which contains an OCCURS clause.

data-name-1
Must contain an OCCURS clause or must be subordinate to a data description entry which contains an OCCURS clause.

integer-1
Can be signed. If signed, it must be positive.

ALL
Used as a function argument for a function that allows a variable number of arguments. Can be used only when the subscripted identifier is used as a function argument and can not be used when condition-name is specified.

data-name-3
Must be a numeric elementary item representing an integer.

Data-name-3 can 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 the index-name.

integer-2, integer-3
Must be an unsigned integer.

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 thought of as a series of nested tables and the most inclusive or outermost table in the nest is considered to be the major table with the innermost or least inclusive table being the minor table, the subscripts are written from left to right in the order major, intermediate, and minor.

For example, if TABLE-THREE is defined as:

```
01  TABLE-THREE.
    05  ELEMENT-ONE OCCURS 3 TIMES.
        10  ELEMENT-TWO OCCURS 3 TIMES.
            15  ELEMENT-THREE OCCURS 2 TIMES    PIC X(8).
```

a valid subscripted reference to TABLE-THREE is:

```
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.

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

- In a USE FOR DEBUGGING statement
- As the subject of a SEARCH statement
- In a REDEFINES clause
- In the KEY IS phrase of an OCCURS clause
- In a LIKE clause

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.

Subscripting Using Integers or Data-Names

When an integer or data-name is used to represent a subscript, it can be used to reference items within different tables. These tables need not have elements of the same size. The same integer or data-name can appear as the only subscript with one item and as one of two or more subscripts with another item. A data-name subscript can be qualified; it cannot be subscripted or indexed. For example, valid subscripted

references to TABLE-THREE—assuming that SUB1, SUB2, and SUB3 are all items subordinate to SUBSCRIPT-ITEM—include:

```
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 allows such operations as table searching and manipulating specific items. To use indexing you associate one or more index-names with 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, which identifies the index-name associated with its table, is an optional part of the OCCURS clause. There is no separate entry to describe the index-name since its definition is completely system dependent. Index-names may be seen as compiler generated registers for the use of this program only. They are not data, or part of the data hierarchy, and must be unique in a COBOL program.

Each index name must follow the rules for formation of a user-defined word.

Each index-name refers to a compiler-generated register or storage area.

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, or
- The SEARCH statement with the ALL phrase, or
- The SET statement.

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 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 for a table element that satisfies a specific condition and to adjust the value of the associated index to indicate that table element.

To be valid during execution, an index value must correspond to a table element occurrence of neither 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. See “INDEXED BY Phrase” on page 170.

Relative Subscripting

In relative subscripting, the name of a table element is followed by a subscript of the form data-name or index-name followed by the operator + or -, and an unsigned integer literal. The operator + and - must be preceded and followed by a space. If the subscript contains a data-name, the value of the subscript is the same as if the data-name had been set up or down by the value of the integer. If the subscript contains an index-name, 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 a USAGE IS INDEX clause. The index value is moved to the index data-item by the execution of a SET statement.

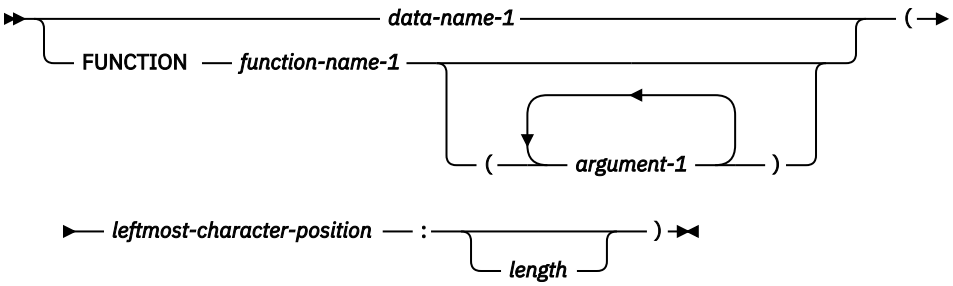
A valid index value must correspond to a table element occurrence of not less than one, nor greater than the highest permissible occurrence number.

Further information on index-names is given in the description of the INDEXED BY phrase of the OCCURS clause. See "INDEXED BY Phrase" on page 170.

Reference Modification

Reference modification resembles the use of 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 data item whose implicit or explicit usage is DISPLAY, DISPLAY-1, or NATIONAL. Data-name-1 can be qualified or subscripted.

[IBM Extension] Must not refer to a Boolean data item. [End of IBM Extension]

[IBM Extension] Must not refer to an item that is defined using the TYPE clause. [End of IBM Extension]

function-name-1
Must be an alphanumeric, DBCS, or national function including its arguments.

For more information, see "Intrinsic Functions" on page 458.

argument-1
Argument-1 must be an identifier, literal (other than a figurative constant), or arithmetic expression.

leftmost-character-position
Must be 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 or function-name-1.

length
Must be an arithmetic expression.

The sum of leftmost-character-position and length minus the value one must be less than or equal to the number of characters in the data item referenced by data-name-1 or function-name-1. If length is omitted, then the length used is equal to the number of characters in the data item referenced by data-name-1 or function-name-1 plus one minus the leftmost-character-position. The evaluation of length must result in a positive nonzero integer.

Note: If the arithmetic expression creates a fixed-point non-integer, truncation occurs, resulting in an integer. If the arithmetic expression creates a floating-point non-integer, rounding occurs, resulting in an integer.

[IBM Extension] For DBCS or national data items, position and length refer to the number of double byte characters. [End of IBM Extension]

Reference modification is generally allowed anywhere an identifier referencing an alphanumeric, DBCS, or national data item is allowed.

[IBM Extension] A data item of class date-time cannot be reference modified. [End of IBM Extension]

Each character of a data item referenced by data-name-1 or function-name-1 is assigned an ordinal number incrementing by one from the left-most position to the right-most position. The left-most 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.

Reference modification creates a unique data item which is a subset of the data item referenced by data-name-1 or by function-name-1 and its arguments. This unique data item is considered an elementary item without the JUSTIFIED clause.

When data-name-1 is reference-modified, the unique data item has the same class and category as that defined for the data item referenced by data-name-1; however, if the category of data-name-1 is numeric, numeric-edited, alphanumeric-edited, or external floating-point, the unique data item has the class and category alphanumeric.

When a function is reference-modified, the unique data item has the class and category of alphanumeric, DBCS, or national depending on the function arguments.

If length is not specified, the unique data item created extends from and includes the character identified by the leftmost-character position up to and including the right-most character of the data item referenced by data-name-1 or function-name-1.

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. If an ALL subscript is specified for an operand, the reference-modifier is applied to each of the implicitly specified elements of the table.
- If reference modification is specified for an intrinsic function, the reference modification is evaluated immediately after evaluation of the function.

Reference Modification Example

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

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

Range Errors

An out-of-range reference modification component, such as a leftmost-character-position of zero, causes system message to be generated. This is the same message that signals errors in subscript ranges and character-string boundaries. (This message is generated only when the RANGE option is specified on the CRTCBLMOD or CRTBNDCBL command.)

Restrictions on Reference Modification

[IBM Extension] The INDICATORS phrase does not support reference modification. [End of IBM Extension]

The following restrictions apply to the statements listed:

Statement
Restriction

STRING
You cannot reference modify identifier-3.

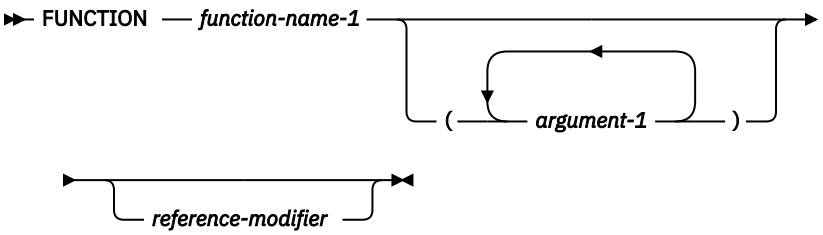
UNSTRING
You cannot reference modify identifier-1.

START
You can reference modify last occurrence of data-name-1 only.

Function-Identifier

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

Format



argument-1
Must be an identifier, literal (other than a figurative constant), or arithmetic expression.

function-name-1
Function-name-1 must be one of the intrinsic function names.

For more information, see "Intrinsic Functions" on page 458.

reference-modifier
Can be specified only for alphanumeric, DBCS, or national functions.

References to Alphanumeric Items

A function-identifier that makes reference to an alphanumeric function can be specified anywhere that an identifier is permitted and where references to functions are not specifically prohibited, except as follows:

- As a receiving operand of any statement
- Where a data item is required to have particular characteristics (such as class and category, size, sign, and permissible values) and the evaluation of the function according to its definition and the particular arguments specified would not have these characteristics.

References to Integers

A function-identifier that makes reference to an integer or numeric function may be used wherever an arithmetic expression is allowed.

[IBM Extension] References to DBCS Items

A DBCS function can be specified anywhere in the general formats that a DBCS identifier is permitted, and where the rules associated with the general formats do not specifically prohibit reference to functions, except as follows:

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

A reference modification for a DBCS function is allowed. If reference modification is specified for a function, the evaluation of the reference modification takes place immediately after the evaluation of the function.

A DBCS function can be referenced as an argument for a function that allows a DBCS argument.

[End of IBM Extension]

[IBM Extension] References to National Items

A national function can be specified anywhere in the general formats that a national identifier is permitted, and where the rules associated with the general formats do not specifically prohibit reference to functions, except as follows:

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

A reference modification for a national function is allowed. If reference modification is specified for a function, the evaluation of the reference modification takes place immediately after the evaluation of the function.

A national function can be referenced as an argument for a function that allows a national argument.

[End of IBM Extension]

[IBM Extension] References to Date-Time Items

A date-time function can be specified anywhere in the general formats that a date-time identifier is permitted, and where the rules associated with the general formats do not specifically prohibit reference to functions, except as follows:

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

A date-time function can be referenced as an argument for a function that allows a date-time argument.

[End of IBM Extension]

[IBM Extension] References to Boolean Items

A boolean function can be specified anywhere in the general formats that a boolean identifier is permitted, and where the rules associated with the general formats do not specifically prohibit reference to functions, except as follows:

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

A boolean function can be referenced as an argument for a function that allows a boolean argument.

[End of IBM Extension]

[IBM Extension] User-Defined Data Types

A user-defined data type (or type name) is a 01 level elementary or group item that contains the TYPEDEF clause. No storage is allocated for such an item. It can be thought of as a template that describes a data name and its subordinate items. A type name can then be used to define a data name (or another type name) by specifying it within a TYPE clause. The defined data name will have the characteristics of the type name specified in the TYPE clause. If the type name is a group item, then the defined data name will be a group item with subordinate items having the same names, hierarchy, and characteristics as the items subordinate to the type name.

When defining a data name (or type name) by using a user-defined data type in a TYPE clause, only the following clauses may be used in conjunction with the TYPE clause to complete the description of the data name:

- EXTERNAL clause
- GLOBAL clause
- OCCURS clause
- TYPEDEF clause
- VALUE clause.

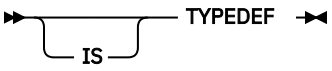
The scoping rules for type names are the same as those for data names.

For more information about the TYPE and TYPEDEF clauses, refer to "TYPE Clause" on page 200 and "TYPEDEF Clause" on page 201.

[End of IBM Extension]

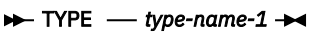
TYPEDEF Clause

The TYPEDEF clause declares an elementary or group data item to be a user-defined data type (or type name). Once the type name has been defined, it can be used (in a TYPE clause) to define other data items.



TYPE Clause

The TYPE clause allows a user-defined data type (or type name) to be used to define a data item. This is done by specifying the type name (which is declared using the TYPEDEF clause) in a TYPE clause. If the type name is a group item, then the defined data item will also be a group item: its subordinate entries will correspond in name, hierarchy, and characteristics to those subordinate to the type name.



type-name-1
The name of the type name that is to be used to define the subject data name.

Scope of Names

This section contains a brief description of the types of COBOL names, followed by the rules for name scoping.

Types of Names

alphabet-name
An alphabet-name assigns a name to a specific character set and/or collating sequence in the SPECIAL-NAMES paragraph of the Environment Division.

class-name
A class-name assigns a name to the preposition in the SPECIAL-NAMES paragraph of the Environment Division for which a truth value can be defined.

condition-name
A condition-name associates a value with a conditional variable.

constant-name
A constant-name identifies a constant, which is defined in the data division.

data-name
A data-name names a data item.

file-name
A file-name names a file connector.

index-name
An index-name names an index associated with a specific table.

library-name
A library-name names a COBOL library that is to be used by the compiler for a given source program compilation.

mnemonic-name
A mnemonic-name assigns a user-defined word to an implementer-name.

paragraph-name
A paragraph-name names a paragraph in the Procedure Division.

program-name
A program-name names a program, either external or internal (nested).

See "Conventions for Program-Names" on page 61.

record-name
A record-name names a record.

section-name
A section-name names a section in the Procedure Division.

symbolic-character
A symbolic-character specifies a user-defined figurative constant.

text-name
A text-name names a library containing source members to be used by the COPY directive statements.

[IBM Extension]

type-name
A type-name names a user-defined data type that can be used in a TYPE clause to define a data item.

[End of IBM Extension]

Nested Programs

A COBOL program may **contain** other COBOL programs. The **contained** (or nested) programs may themselves contain yet other programs. A contained program may be **directly** or **indirectly** contained within a program.

Figure 2 on page 58 describes a program structure with directly and indirectly contained programs.

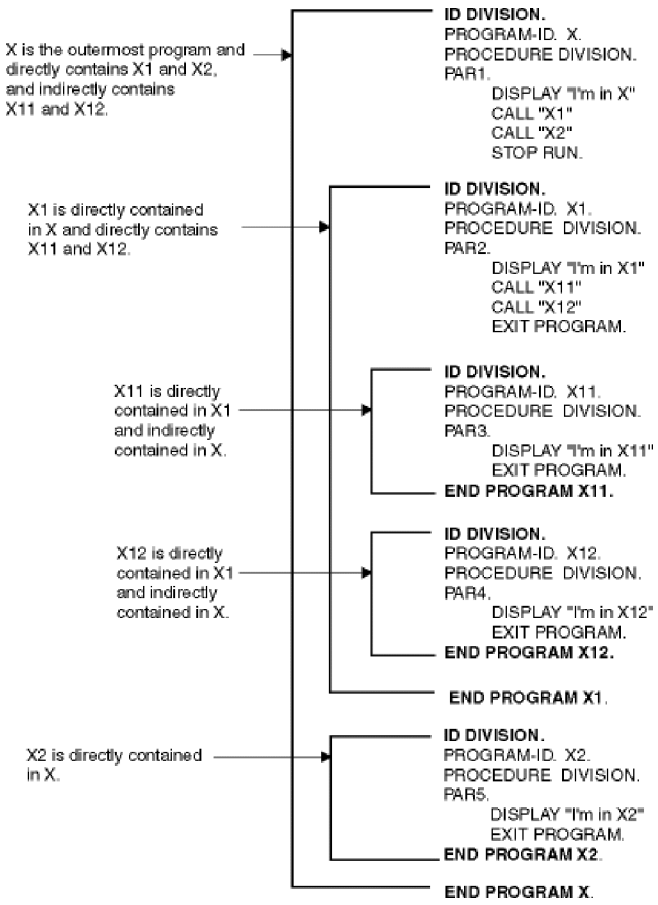


Figure 2. Nested Program Structure with Directly and Indirectly Contained Programs

The same user-defined word may be used in different programs to define different objects. In a particular program, a reference to an object always refers to the object defined **in that program**.

Global and Local Names

Names can have global or local attributes. Some names are always global; other names are always local; and some names are either local or global depending upon specifications in the program in which the names are declared.

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

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

A **local name**, however, may be used only to refer to the object with which it is associated from within the program in which the local name is declared.

If a data-name, record-name, condition-name, type-name, or file-name is not declared to be global, the name is local.

Note: Specific rules sometimes prohibit specification of the GLOBAL clause for certain data description, file description, or record description entries.

constant-name
A constant-name is global if the GLOBAL clause is specified.

data-name
A data-name is global if the GLOBAL clause is specified either in the data description entry by which the data-name is declared or in another entry to which that data description entry is subordinate.

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

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

1. An external file connector can be referenced from any program that describes that file connector.
2. If a program is contained within another program, both programs can refer to a common file connector by referring to an associated global file-name declared either in the containing program or in any program that directly or indirectly contains the containing program.

record-name
A record-name is global if the GLOBAL clause is specified in the record description entry by which the record-name is declared or, in the case of record description entries in the File Section, if the GLOBAL clause is specified in the file description entry for the file-name associated with the record description entry.

condition-name
A condition-name, when declared in the data description entry, is global if that entry is subordinate to another entry in which the GLOBAL clause is specified.

A condition-name, when declared within the Configuration Section, is always global.

program-name
A program-name is neither local nor global. See "Conventions for Program-Names" on page 61.

section-name and paragraph-name
These names are always local.

library-name and text-name
These names are external to the program and can be referenced by any COBOL program, provided that the compiler system supports the associated library and the entities referenced are known to that system.

alphabet-name
An alphabet-name is always global.

class-name
A class-name is always global.

mnemonic-name
A mnemonic-name is always global.

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

[IBM Extension]

type-name
A type name is global if the GLOBAL clause is specified in the data description entry by which the type-name is declared. The GLOBAL attribute of a type-name is restricted to the type-name, and is not acquired by a data item that is defined using the type-name in a TYPE clause.

[End of IBM Extension]

External and Internal Objects

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

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

programs using separate descriptions of the object with the same name are always to the same object. In a run unit, there is only one representative of an external object.

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

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

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

A file connector is given the external attribute by the presence of the EXTERNAL clause in the associated file description entry. If the file connector does not have the external attribute, it is internal to the program in which the associated file-name is described. The EXTERNAL clause cannot be specified for sort-merge files.

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

Data records and subordinate data items described in the Linkage Section of a program are always considered to be internal to the program describing that data. Special considerations apply to data described in the Linkage Section whereby an association is made between the data records described and other data items accessible to other programs.

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.

Resolution of Names

When a program, program B, is directly or indirectly contained within another program, program A, both programs may define objects using the same user-defined word. (Objects include, for example, a condition-name, a data-name, a file-name, a record-name, a function name, or a type-name.) When such a duplicated name is referenced in program B, the following rules are used to determine the referenced object:

1. The referenced object is identified from the set of all names which are defined in program B and all global names defined in the directly containing program A and in any programs which directly or indirectly contain program A. Using this set of names, the normal rules for qualification and any other rules for uniqueness of reference are applied until one or more objects is identified.
2. If only one object is identified, it is the referenced object.
3. If more than one object is identified, no more than one of them can have a name local to program B unless each reference to them can be made unique with appropriate qualification. If zero or one of the objects has a name local to program B, the following rules apply:
   - If the name is declared in program B, the object in program B is the referenced object.
   - Otherwise, if program A is contained within another program, the referenced object is:
     – The object in program A if the name is declared in program A.

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

- When the referenced object is a function, the function definition sometimes requires the programmer to specify a value or set of values for one or more arguments that determine the value of the function for that particular reference. The term **function-identifier** refers to the term used to reference an intrinsic function within the Procedure Division of a COBOL source program. The data item represented by a function is uniquely identified by a function-name with its arguments, if any.

Conventions for Program-Names

The program-name of a program is specified in the PROGRAM-ID paragraph of the program's Identification Division. A program-name can be referenced only by the:

- CALL statement
- CANCEL statement
- END PROGRAM header
- [IBM Extension] SET statement [End of IBM Extension]

Names of programs constituting a run unit are not necessarily unique, but when two programs in a run unit are identically named, at least one of those two programs must be directly or indirectly contained within another separately compiled program which does not contain the other of those two programs.

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

- If the program-name is that of a program which does not possess the COMMON attribute and which is directly contained within another program, that program-name can be referenced only by statements included in that containing program.
- If the program-name is that of a program which possesses the COMMON attribute and that is directly contained within another program, that program-name can be referenced only by statements included in that containing program and any programs directly or indirectly contained within that containing program, except that program possessing the COMMON attribute and any programs contained within it.
- If the program-name is that of an outermost COBOL program in a separately compiled module object, that program-name can be referenced by statements included in any other program in the run unit, except programs it directly or indirectly contains.

Rules Regulating the Scope of Program Names

The following rules apply to referencing a program-name of a program that is contained within another program. For this discussion, we will say that Program-A directly contains Program-B and Program-C, Program-C directly contains Program-D and Program-F, and Program-D directly contains Program-E.

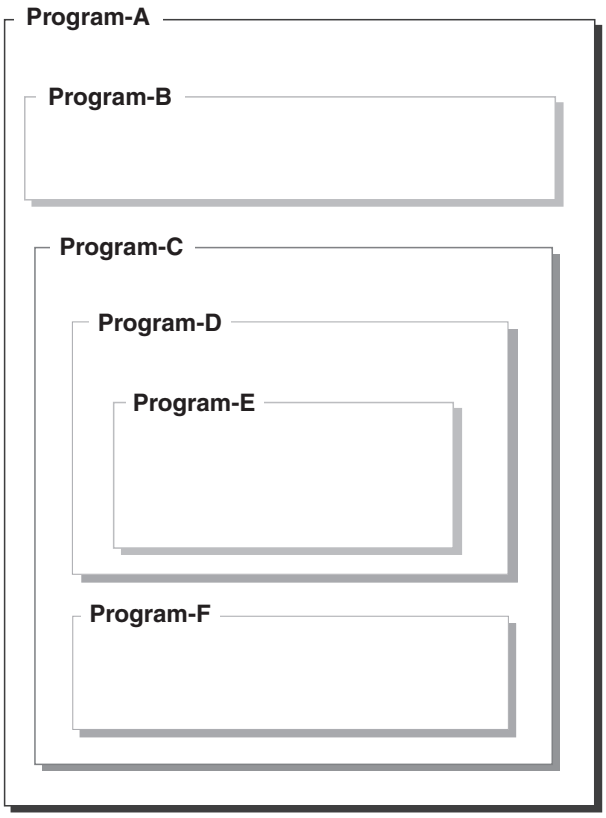


Figure 3. Rules Regulating the Scope of Program Names

If Program-D does not possess the COMMON attribute, then Program-D can only be referenced by the program that directly contains Program-D, that is, Program-C.

If Program-D does possess the COMMON attribute, then Program-D can be referenced by Program-C since it contains Program-D, and by any programs contained in Program-C except for Program-D and programs contained in Program-D. In other words, if Program-D possesses the COMMON attribute, Program-D can be referenced in Program-C and Program-F, but not by statements in Program-E, Program-A, Program-B, or Program-D.

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 COBOL procedure being executed under control of another COBOL statement, control implicitly transfers. (COBOL statements that control COBOL 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.

- When a program that has no procedure division or any nondeclarative sections is called, the calling program issues an implicit EXIT PROGRAM.

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

Next Executable 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.
- Following the END PROGRAM header.

When there is no next executable statement and control isnot 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.

Chapter 3. COBOL Program Structure

General Structure

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

A COBOL source program may contain other COBOL source programs. These contained programs may reference some of the resources of the programs within which they are contained.

This concept of contained programs is known as nesting and the contained program is known as a **nested program**. A nested program may be directly or indirectly contained in another program. For example, if program B is contained in Program A, it is directly contained if there is no program in program A that also contains program B. Program B is indirectly contained in program A if there exists a program contained in Program A that also contains program B.

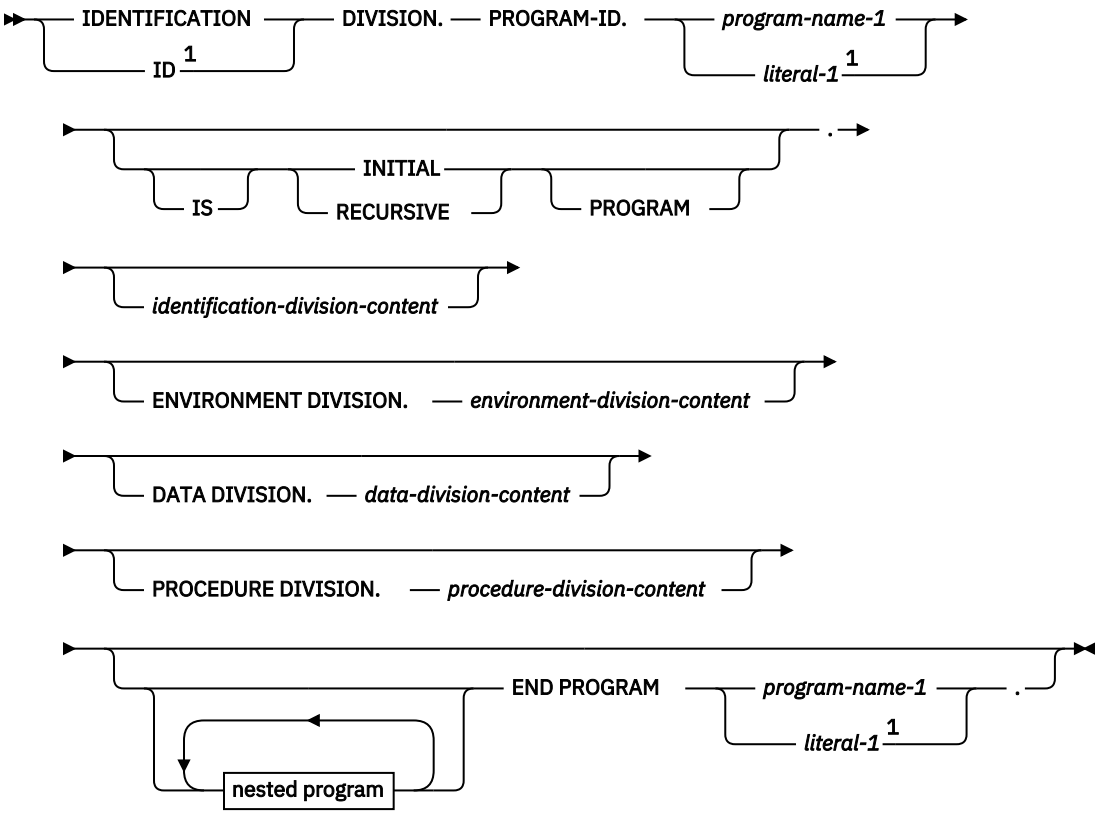
For more information on contained and containing programs, refer to the section on nested programs in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

With the exception of compiler-directing statements and the END PROGRAM header, the statements, entries, paragraphs, and sections of a COBOL source program are grouped into the following four divisions:

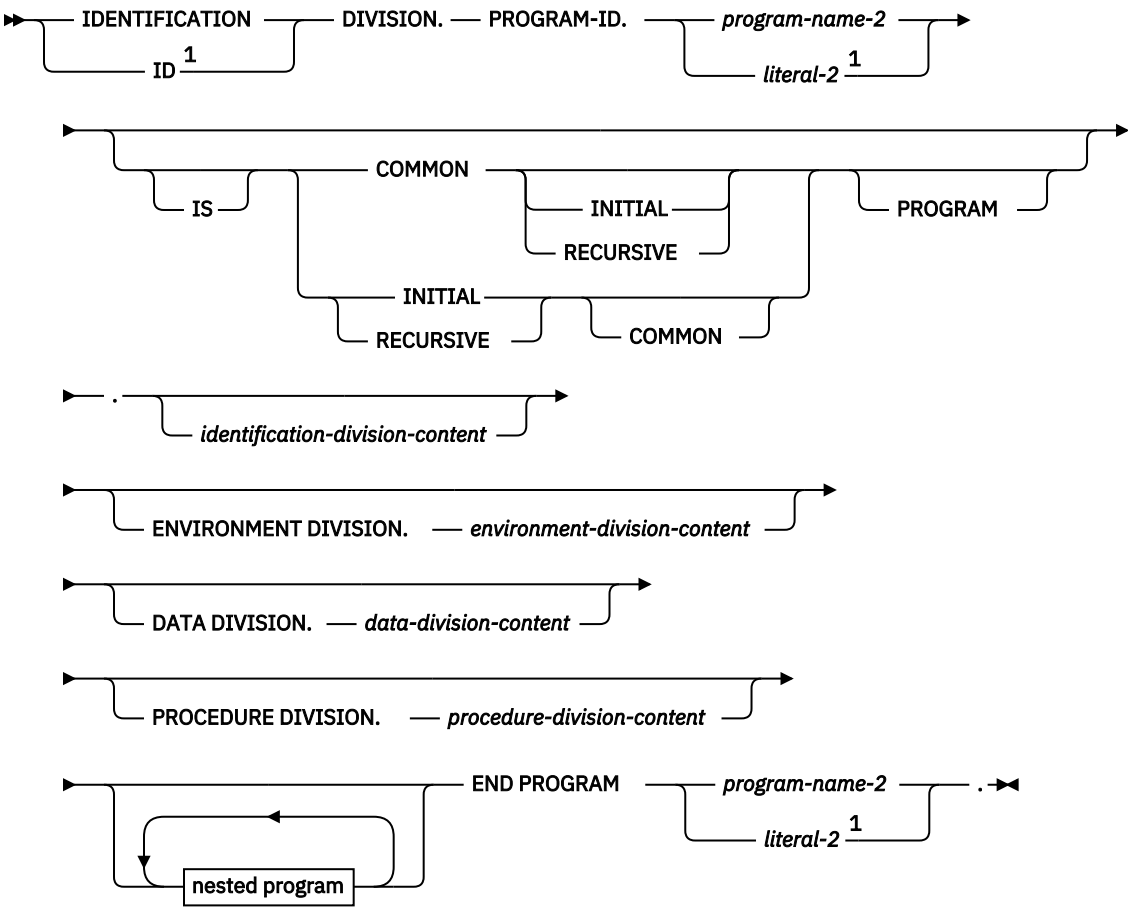
- Identification Division
- Environment Division
- Data Division
- Procedure Division

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

The following figure briefly illustrates the general structure of a COBOL program.



nested program

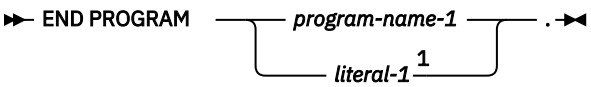


COBOL Source Program—Format
Notes:

[1] IBM Extension

END PROGRAM Header

The END PROGRAM header indicates the end of a named COBOL source program. It also separates each program in a sequence of source programs. An END PROGRAM header is optional for the last program in a sequence of source programs only if that program does not contain any nested programs.



END PROGRAM Header - Format
Notes:

[1] IBM Extension

program-name-1
A user-defined word that must be identical to a program-name declared in a preceding PROGRAM-ID paragraph. Refer to *program-name* in "PROGRAM-ID Paragraph" on page 70 for the rules for formation of the program-name.

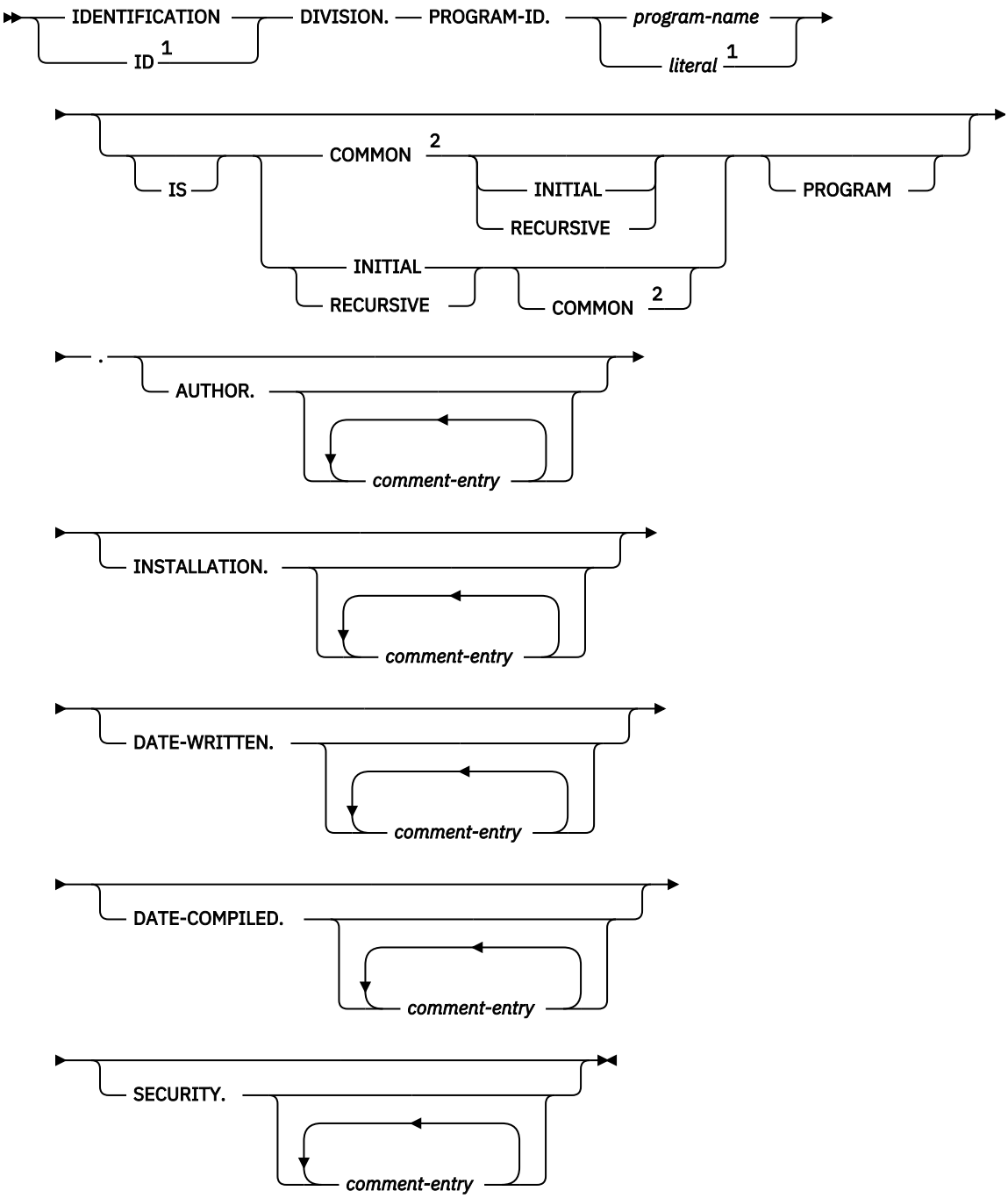
[IBM Extension]

literal-1
Must be a nonnumeric literal. Refer to *literal* in "PROGRAM-ID Paragraph" on page 70 for the rules for formation of the literal.

[End of IBM Extension]

Chapter 4. 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.



Identification Division - Format

Notes:

[1] IBM Extension

[2] Allowed only for nested COBOL programs

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.

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

Figure 4 on page 70 shows how the coding for the Identification Division should look. [End of IBM Extension]

```
IDENTIFICATION DIVISION.
PROGRAM-ID. IDSAMPLE.
AUTHOR. PROGRAMMER NAME.
INSTALLATION. COBOL DEVELOPMENT CENTER.
DATE-WRITTEN. 12/02/94.
DATE-COMPILED. 12/09/94 12:57:53.
SECURITY. NON-CONFIDENTIAL.
```

Figure 4. Identification Division Coding Example Showing Sentences Beginning on Same Lines as Paragraphs

PROGRAM-ID Paragraph

The PROGRAM-ID paragraph specifies the name of the COBOL program. For an outermost program, it can also specify the name of the program object (*PGM) or module object (*MODULE), or both. It is required and must be the first paragraph in the Identification Division.

The name by which the program object is known to the system can be overridden by the PGM parameter of the CRTBNDCBL command. The name by which the module object is known can be overridden by the MODULE parameter of the CRTCBLMOD command. See the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information on the PGM or MODULE parameter.

program-name

A user-defined word that identifies your program or module object to the system. For program and module objects, only the first 10 characters of program-name are used as the identifying name of the object. For an ILE procedure name, the first 250 characters of program-name are used. If the *MONOPRC option is specified on the CRTBNDCBL or CRTCBLMOD command, 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).

When *PGMID is specified on the CRTBNDCBL or CRTCBLMOD command for the program or module name and the *NOMONOPRC option is specified, the user must ensure that the program name specified in the

PROGRAM-ID paragraph does not contain a non-numeric literal containing lowercase characters as this may create unusable objects. Care should be taken for multiple source programs where, even in the absence of *PGMID for the program or module name, unusable objects may be created for the second and subsequent ILE COBOL procedures containing non numeric literal with lowercase characters for the PROGRAM-ID paragraph when SIMPLEPGM = *YES.

literal

Must be a nonnumeric literal.

A nonnumeric literal without the enclosing delimiters becomes the program-name. The same rules apply for forming module, program, and procedure names as defined above under program-name. If the *MONOPRC option is specified, however, lowercase letters in the literal are converted to their uppercase equivalents.

[IBM Extension] RECURSIVE Clause

The RECURSIVE clause is an optional clause that allows COBOL programs to be recursively re-entered. This clause specifies that the program and any program contained within it are recursive. ILE COBOL allows the RECURSIVE clause in a nested program. As well, recursive programs will be able to contain a nested subprogram. Program-name-1 can be recursively re-entered while a previous invocation is still active if the RECURSIVE clause is specified. An active program cannot be recursively re-entered if the RECURSIVE clause is not specified.

The Working-Storage Section of a recursive program defines storage that is statically allocated and initialized on the first entry to a program, and is available in a last-used state to any of the recursive invocations. The Local-Storage Section of a recursive program (as well as a non-recursive program) defines storage that is automatically allocated, initialized, and deallocated on a per-invocation basis.

Internal file connectors corresponding to FDs in the File Section of a recursive program are statically allocated. The status of internal file connectors is part of the last-used state of a program that persists across invocations.

The following language elements are not supported in a recursive program:

- ALTER; see the “ALTER Statement” on page 277
- GO TO without a specified procedure name; see the “GO TO Statement” on page 320
- RERUN; see the “RERUN Clause” on page 118
- SEGMENTATION
- USE FOR DEBUGGING; see “USE FOR DEBUGGING” on page 535

The RECURSIVE clause shall not be specified if any program that directly or indirectly contains this program is an inital program.

[End of IBM Extension]

COMMON Clause

The COMMON clause allows the program named by program-name to be called by its [1] and by programs contained within the siblings. The COMMON clause can be used only in nested programs.

INITIAL Clause

Specifies that when program-name is called, program-name and any programs contained within it are set to their initial state. (All working storage items are reset to their initial values and all INTERNAL files are closed.)

A program is set to its initial state:

[1] Sibling programs are those that are directly contained by the same program.

- The first time the program is called in a run unit
- Every time the program is called, if it possesses the INITIAL attribute
- The first time the program is called after the execution of a CANCEL statement referencing the program or a CANCEL statement referencing a program that directly or indirectly contains the program
- The first time the program is called after the execution of a CALL statement referencing a program that possesses the INITIAL attribute, and that directly or indirectly contains the program.

  For example, if program A calls program B, and program B has the INITIAL attribute and also contains program C, program C will be set to its initial state the first time that it is called after A called B.

When a program is set to its initial state, the following occur:

- The program's internal data contained in the Working-Storage and Local-Storage sections are initialized. If a VALUE clause is used in the description of the data item, the data item is initialized to the defined value. If a VALUE clause is not associated with a data item, the initial value of the data item depends on whether the *STDINZ, *STDINZHEX00 or the *NOSTDINZ option is specified on the CRTCBLMOD or CRTBNDCBL command.
- Files with internal file connectors associated with the program are not in the open mode.
- The control mechanisms for all PERFORM statements contained in the program are set to their initial states.
- The altered GO TO statements contained in the program are set to their initial state.

The INITIAL clause shall not be specified if any program that directly or indirectly contains this program is a recursive program.

Optional Paragraphs

These optional paragraphs in the Identification Division may be omitted.

AUTHOR
Name of the author of the program. It is syntax checked only.

INSTALLATION
Name of the company or location. It is syntax checked only.

DATE-WRITTEN
Date the program was written. It is syntax checked only.

DATE-COMPILED
Date the program was compiled.

SECURITY
Level of confidentiality of the program.

comment-entry

The **comment-entry** in any of the optional paragraphs may be any combination of characters from the character set of the computer. Do not confuse the comment-entry with a comment line. (The latter is indicated by a slash or asterisk in the indicator area.) The comment-entry is written in Area B on one or more lines. The SEU Syntax Checker, however, requires that the first sentence begin on the same line as the paragraph header.

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.

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.
```

The first eight lines of the comment-entry in the SECURITY paragraph will form the copyright information in the created module object.

[IBM Extension]

A 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.

Comments may combine double-byte and single-byte character-strings.Multiple lines are allowed in a comment-entry containing double-byte strings, however shift-out and shift-in characters must be paired in a line.

Note: Mixed strings are described under “Character-Strings” on page 24.

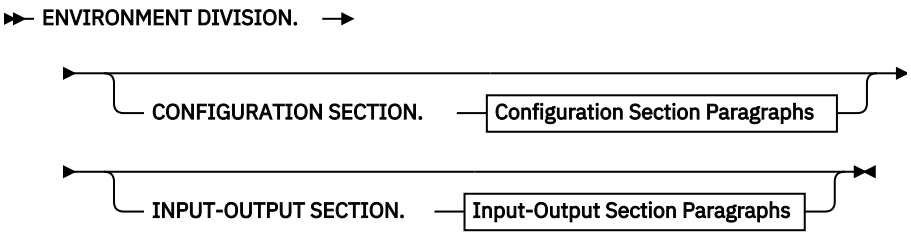
[End of IBM Extension]

Chapter 5. Environment Division

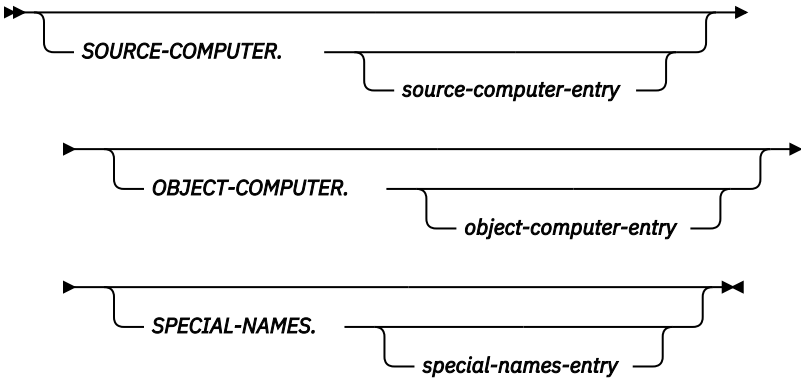
The Environment Division has two sections:

- The Configuration Section
- The Input-Output Section. (See “Input-Output Section” on page 95.)

The Environment Division is optional in a COBOL source program.

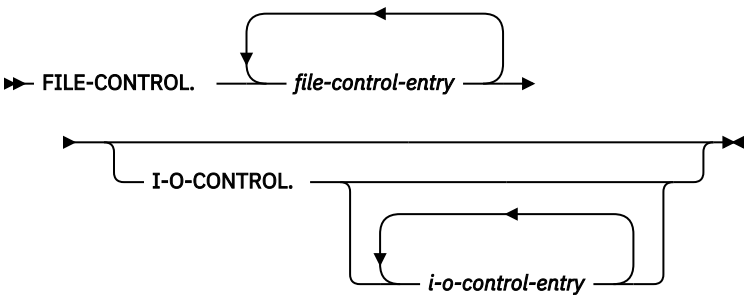


Configuration Section Paragraphs



Environment Division - Format

Input-Output Section Paragraphs



Configuration Section

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. However, the Configuration Section must not be specified in a nested program. The entries specified in the Configuration Section of a program apply to all programs contained within that program.

In addition, the Configuration Section can:

- Relate IBM-defined environment-names to user-defined mnemonic names
- Specify the collating sequence

- Specify a single or multiple character currency string and a substitute character 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
- Specify the type of linkage to be made on a CALL, CANCEL, or SET...ENTRY statement
- Specify the default formats for a date or time data type.

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

The Configuration Section of the Environment Division contains three paragraphs:

- SOURCE-COMPUTER paragraph
- OBJECT-COMPUTER paragraph
- SPECIAL-NAMES paragraph.

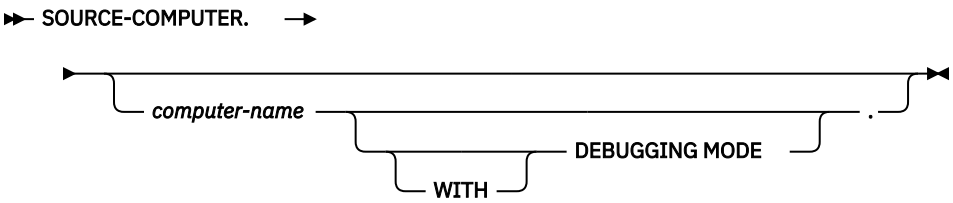
Coding Example

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-I.
OBJECT-COMPUTER. IBM-I.
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.

SOURCE-COMPUTER Paragraph - Format



computer-name

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

```
IBM-I
```

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

WITH DEBUGGING MODE Clause

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.

To specify a debugging line in your program, code a 'D' or 'd' in column 7 (indicator area). You may include successive debugging lines, but each must have a 'D' or '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 determined after all COPY statements are processed. See "COPY Statement" on page 508 for details.

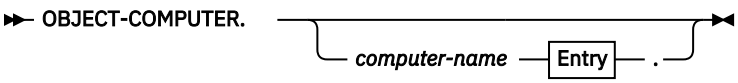
You may code debugging lines in the Environment (after the OBJECT-COMPUTER paragraph), Data, or Procedure Divisions.

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

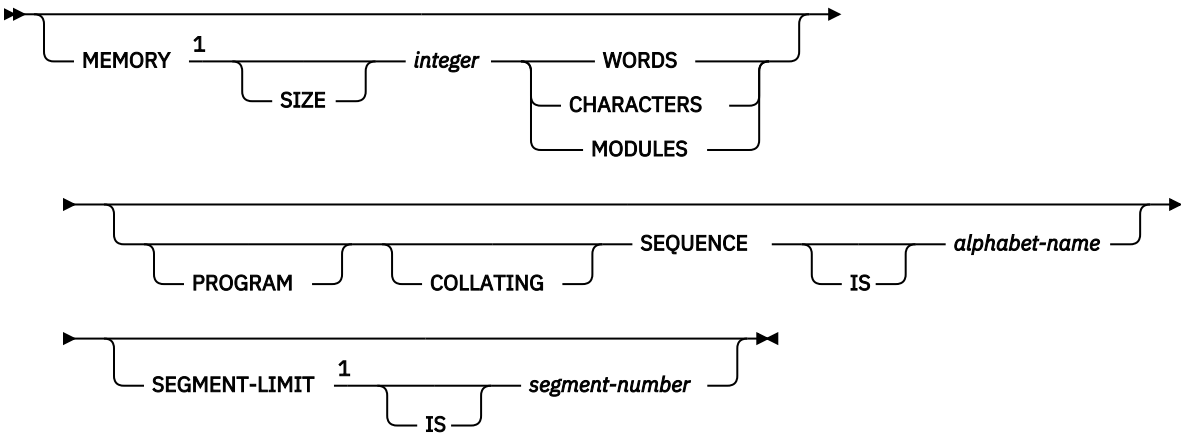
If the WITH DEBUGGING MODE clause is omitted, debug lines are treated as comment lines.

OBJECT-COMPUTER Paragraph

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



Entry



OBJECT-COMPUTER Paragraph - Format

Notes:

[1] Syntax-checked only.

computer-name

A system-name, which is syntax checked but has no effect on the execution of the program. The suggested computer-name is:

```
IBM-I
```

MEMORY SIZE

The amount of main storage needed to run the object program. The MEMORY SIZE clause is syntax checked only.

integer
Expressed in words, characters, or modules.

PROGRAM COLLATING SEQUENCE IS
The collating sequence used in this program (and any programs it may contain) 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 C. EBCDIC and ASCII Collating Sequences" on page 546 for more information about these sequences.

SEGMENT-LIMIT IS
Determines which segments will be considered as permanent segments of the object program. This clause is syntax checked only.

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

SPECIAL-NAMES Paragraph

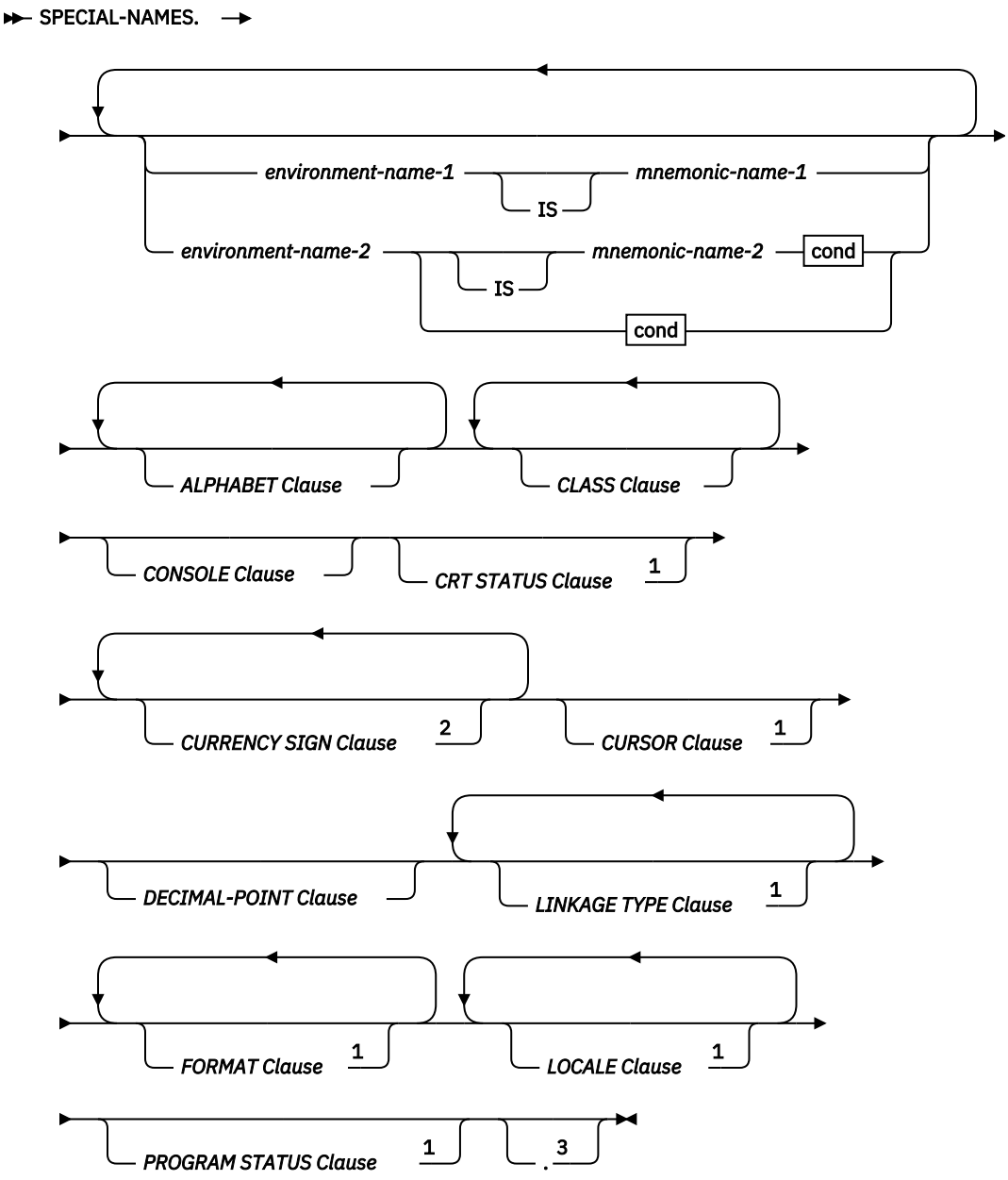
The SPECIAL-NAMES paragraph:

- 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 single or multiple character currency string and 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.

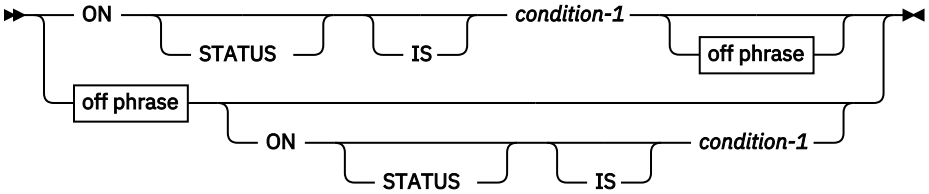
[IBM Extension]

- Relates locale object names and their associated library to user-defined mnemonic-names.
- Specifies the default formats for a date or time data type.
- Specifies that ACCEPT or DISPLAY statements are treated as extended ACCEPT or DISPLAY statements, or as requests to the dynamic screen manager session services APIs.
- Specifies additional functions associated with ACCEPT statements.
- Specifies the type of linkage used for a CALL or CANCEL of a program, and for setting a procedure-pointer with the SET statement.

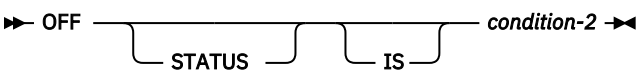
[End of IBM Extension]



cond



off phrase



SPECIAL-NAMES Paragraph - Format

Notes:

[1] IBM Extension

[2] Subsequent repetitions are IBM Extensions.

[3] The separator period must be used if any of the optional clauses are selected. Clauses can be entered in any order.

environment-name-1
System devices or standard system actions taken by the compiler.

Table 2 on page 80 shows the actions that are associated with mnemonic-names for environment-name-1.

Table 2. 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 the device is PRINTER.
C01	WRITE	Skip to the next page. Use only when the device is PRINTER.
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.
DATA-AREA	ACCEPT, DISPLAY	Retrieves or updates a system data area.
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.
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).

environment-name-2
Environment-name-2 can be defined as UPSI-0 through UPSI-7 or as SYSTEM-SHUTDOWN; UPSI stands for a one-byte User Programmable 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 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 condition-1 or condition-2 names.

Mnemonic-names and environment-names need not be unique. If you choose a mnemonic-name that is also an environment name, its definition as a mnemonic-name takes precedence over its definition as an environment-name for a given reference to such a name.

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 236.)

condition-1, condition-2

If environment-name-2 references an external switch, the on/off status of that switch can be associated with condition-names, such as condition-1, condition-2. The status of the switch can be obtained through the condition-names. 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.

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.

Any names declared in a program's SPECIAL-NAMES paragraph can be referenced from any contained program.

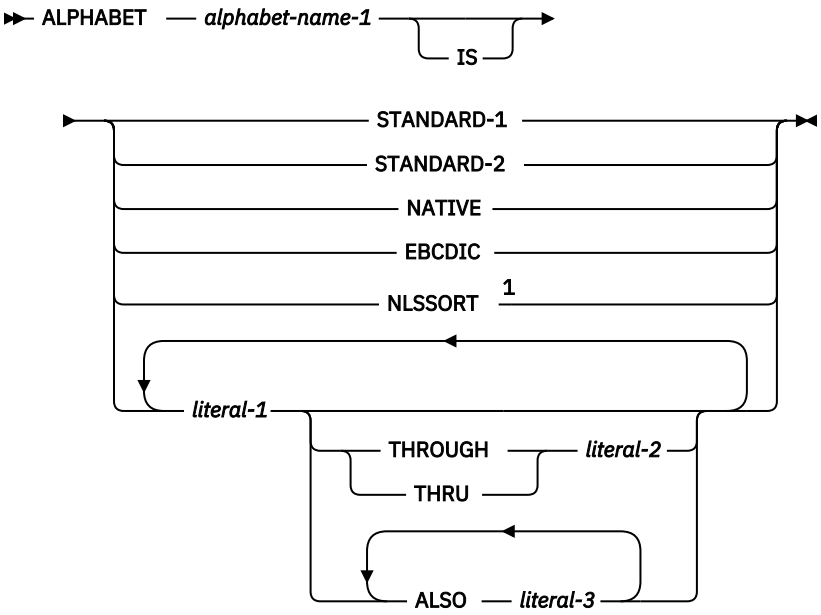
Coding Example

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

The ALPHABET clause provides a means of relating an alphabet-name to a specified character code set or collating sequence.



ALPHABET Clause - Format

Notes:

[1] IBM Extension

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.

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

- ALPHABET Clause Coding Examples.

alphabet-name-1
Alphabet-name-1 follows the rules for user-defined names. At least one character must be alphabetic. Alphabet-name-1 identifies a specific character code set or collating sequence.

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 EBCDIC character set.

EBCDIC
Specifies the EBCDIC character set.

NLSSORT
Use the SRTSEQ and LANGID specifications in the compiler options (or implied defaults) for alternate collating sequence aspects of the alphabet-name. An alphabet-name associated with NLSSORT can

be referred to only in the PROGRAM COLLATING SEQUENCE clause or in the COLLATING SEQUENCE phrase of the SORT and MERGE statements.

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 C. EBCDIC and ASCII Collating Sequences" on page 546, 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 (if it is longer, only the first character is kept, and a warning is issued)
- 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 149).

[IBM Extension] DBCS literals and floating-point literals may not be used in a user-specified collating sequence. [End of IBM Extension]

Coding Examples

The following examples illustrate some uses for the ALPHABET 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:

```
77 ITEM-1 PIC X(3) VALUE "ABC".
77 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:

```
77 ITEM-1 PIC X(3) VALUE HIGH-VALUE.
77 ITEM-2 PIC X(3) VALUE "789".
```

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:

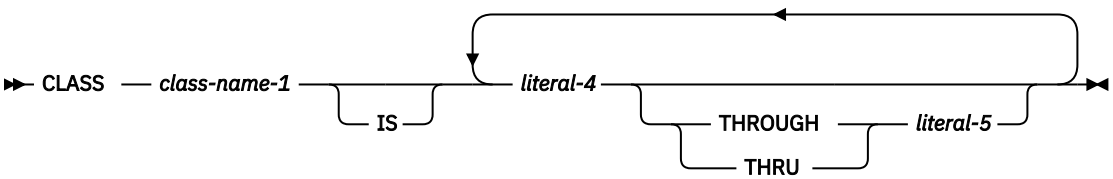
```
05 TABLE A OCCURS 6 ASCENDING KEY IS
   KEY-A INDEXED BY INX-A.
   10 FIELD-A ...
   10 KEY-A ...
```

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

The CLASS clause relates a name to the specified set of characters listed in that clause.

CLASS Clause - Format



class-name-1

Class-name-1 is a user-defined word and must contain at least one alphabetic character. The class-name in the CLASS clause can be a DBCS user-defined word. Class-name-1 can be referenced only in

a class condition. See “Class Condition” on page 225 for more information. The characters specified by the values of the literals in this clause define the exclusive set of characters of which class-name-1 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 C. EBCDIC and ASCII Collating Sequences” on page 546 lists the ordinal number for each character in the EBCDIC collating sequence.

[IBM Extension] Cannot be specified as floating-point literals, DBCS literals, or national literals. [End of IBM Extension]

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.

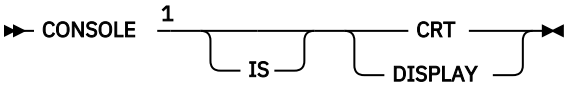
If the nonnumeric literal is associated with a THROUGH phrase, it must be one character in length.

THROUGH, THRU
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.

[IBM Extension] CONSOLE Clause

If CONSOLE IS CRT is specified, 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.

Similarly, if CONSOLE IS DISPLAY is specified, any ACCEPT or DISPLAY statement that has no phrases specific to a particular format is treated as a request to the dynamic screen manager session services APIs. For information on these APIs, see the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.



CONSOLE Clause - Format
Notes:

[1] IBM Extension

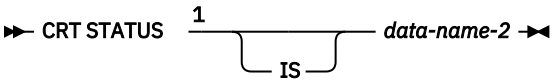
If no CONSOLE IS clause is specified, any ACCEPT or DISPLAY statement that has no phrases specific to a particular format is treated as a standard ANSI COBOL ACCEPT or DISPLAY statement.

See the “Extended ACCEPT and Extended DISPLAY Considerations” on page 271 and the “Format 3 – Extended DISPLAY Statement” on page 303 for descriptions of the conditions which determine whether ACCEPT or DISPLAY statements are extended or standard.

[End of IBM Extension]

[IBM Extension] CRT STATUS Clause

The CRT STATUS clause specifies a data item into which a status value is moved after an extended ACCEPT statement.



CRT STATUS Clause - Format

Notes:

[1] IBM Extension

data-name-2
Must be described in the WORKING-STORAGE or LOCAL-STORAGE SECTIONS and must be a 6-byte alphanumeric field or a 6-byte unsigned zoned integer. If data-name-2 is referenced from a nested program, it must be defined as global in the outermost program.

[End of IBM Extension]

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

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 3. 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 4 on page 87.

Table 4. Auto Skip Value Returned by Controller Type	
Type of Controller	**Auto Skip Value of 1 Returned**
IBM i controllers: Local workstation controllers Remote 5251 model 12 Remote 5294 Remote 5394 Remote 3174 Remote 3274	 Yes Not applicable No Yes, if installed with new workstation controller code No, with *NOUNDSPCHR option No, with *NOUNDSPCHR option
PC attachments: DOS and Operating System/2® (OS/2) operating environments	No
System to system passthru: IBM i system to IBM i system System/36™ to IBM i system System/38™ to IBM i system	 Yes No 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
- 93 Help key
- 94 Clear key

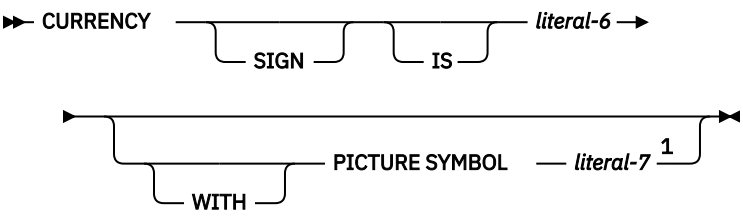
Help and Clear keys accept data only on local IBM i workstations.

CURRENCY SIGN Clause

The CURRENCY SIGN clause is used to define a **currency string** that will be:

- Inserted into a numeric-edited data item when it is used as a receiving item
- Removed from a numeric-data item (de-edited) when determining the unedited numeric value of the item.

In addition, the clause may also be used to specify the symbol that is to be used to represent a currency string within a PICTURE character-string. This symbol is referred to as the **currency symbol**.



CURRENCY SIGN Clause - Format

Notes:

[1] IBM Extension

[IBM Extension]

Note: The CURRENCY SIGN clause can be repeated to allow for more than one currency string in a COBOL program. However, the value of a currency symbol must **not** be duplicated.

[End of IBM Extension]

When the CURRENCY SIGN clause is omitted, the dollar sign ($) must be used for both the value of the currency string and the currency symbol.

literal-6 without PICTURE SYMBOL phrase

Specifies the value of the **currency string** as well as the character that will be used as the **currency symbol**. It must be a single-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, or their lower case equivalents
- A space
- Special characters * + - / , . ; ( ) = "
- A figurative constant

[IBM Extension]

- The uppercase alphabetic character E if the program defines an external floating-point item.
- Uppercase alphabetic characters G and N if the program defines a DBCS or national item.
- Lowercase alphabetic characters e, g, and n.

[End of IBM Extension]

The currency symbol is case sensitive and must be specified throughout your program with the same case as used in the CURRENCY SIGN clause. However, unless the OPTION parameter value *NOMONOPIC, or the PROCESS statement option NOMONOPIC is specified, an alphabetic currency symbol used in a PICTURE character-string will be considered to be uppercase, regardless of its actual representation. Therefore an alphabetic currency symbol must always be entered as uppercase, unless the NOMONOPIC option is specified.

literal-6 with PICTURE SYMBOL phrase

If the PICTURE SYMBOL phrase is specified, literal-6 specifies the value of the **currency string** and literal-7 represents the currency symbol. Literal-6 may have any length (multiple characters) and may consist of any characters from the computer's character set **except for** the following:

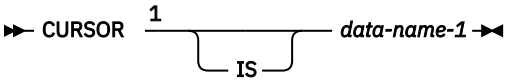
- Digits zero (0) through nine (9)
- Special characters * + - / . ,
- A space or spaces without any other characters

literal-7

If the PICTURE SYMBOL phrase is specified, literal-7 specifies the character that will be used as the **currency symbol**. It must be a single-character, nonnumeric literal, and must **not** have the same value as any other currency symbol defined in the program. The value of this character is subject to the same restrictions as those that apply to the currency sign (literal-6) when the PICTURE SYMBOL phrase is omitted.

[IBM Extension] CURSOR Clause

The CURSOR clause specifies the data item that will contain the cursor address used by the extended ACCEPT statement.



CURSOR Clause - Format

Notes:

[1] IBM Extension

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 described in the WORKING-STORAGE or LOCAL-STORAGE SECTIONS. If data-name-1 is referenced from a nested program, it must be defined as global in the outermost program.

[End of IBM Extension]

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.

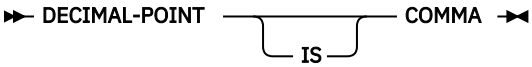
If the CURSOR IS identifier contains an invalid value (such as spaces, low-values, high-values or a value outside of the screen range), the cursor is positioned at the start of the first input field that is active on the screen.

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

DECIMAL-POINT IS COMMA Clause

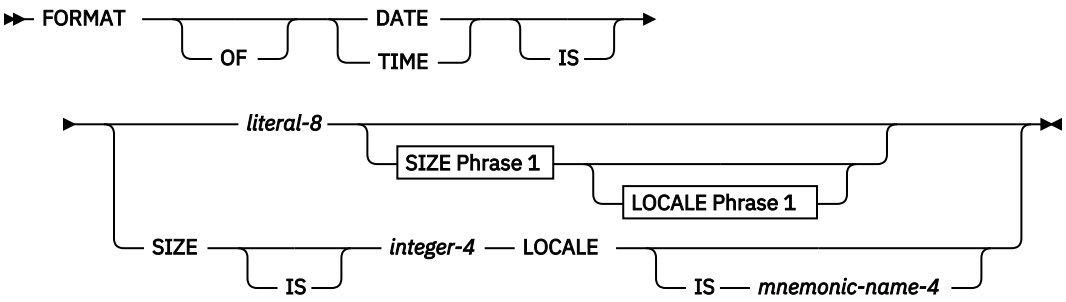
The DECIMAL-POINT IS COMMA clause exchanges the functions of the period and the comma in PICTURE character strings and in numeric literals.

DECIMAL-POINT IS COMMA Clause - Format

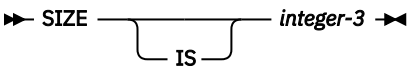


[IBM Extension] FORMAT Clause

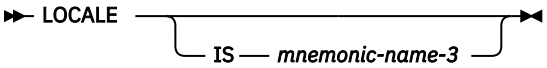
The FORMAT clause is used to specify a default format for a DATA DIVISION date or time item. The format clause can also specify the default date or time format for an intrinsic function.



SIZE Phrase 1



FORMAT Clause - Format
LOCALE Phrase 1



literal-8

Specifies the default format of a date or time item. Literal-8 must be a nonnumeric literal at least 2 characters in length. Literal-8 must contain one or more conversion specifiers and zero or more separators. For more information about the effects of literal-8 on the LOCALE phrase, refer to "LOCALE Phrase" on page 92. For a list of the conversion specifiers that can be used in literal-8, refer to Table 5 on page 90.

The following rules apply:

- When no LOCALE phrase is specified with literal-8, the conversion specifications are replaced with values based on the COBOL locale. For more information about the COBOL locale, refer to *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.
- For a date item, literal-8 must contain a conversion specifier that will result in the day of the year. If literal-8 contains a year and month conversion specification, but no day conversion specification, the first day of the month is assumed. For a list of IBM i date formats and their literal-8 equivalents, refer to Table 10 on page 161.
- If no FORMAT clause is specified for a date item, the default date item format is ISO.
- If literal-8 is not specified, the LOCALE phrase must be specified.
- For a time item, literal-8 must contain an hour and minute conversion specification. If no seconds (or milliseconds) are specified, a value of 0 is assumed. For a list of IBM i time formats and their literal-8 equivalents, refer to Table 11 on page 161.
- If no FORMAT clause is specified for a time item, the default time item format is ISO.

Table 5 on page 90 lists the conversion specifiers that can be used in literal-8.

Table 5. Conversion Specifiers that Can Be Used in Literal-8

Specifier	Description	Length	Allowed For
@C	Replaced by the century as an integer [0,9] (0⁴ 20th century)	1 bytes	D
%d	Replaced by the day of month as an integer [01,31]	2 bytes	D
%D	Same as %m/%d/%y	8 bytes	D
%H	Replaced by the hour (24-hour clock) as an integer [00,23]	2 bytes	T
%I	Replaced by the hour (12-hour clock) as an integer [01,12]	2 bytes	T
%j	Replaced by the day of the year as an integer [001,366]	3 bytes	D
%m	Replaced by the month as an integer [01,12]	2 bytes	D
%M	Replaced by the minute as an integer [00,59]	2 bytes	T
%p	Replaced by the locale's equivalent of either a.m. or p.m.	locale	T
@p	AM and PM can be any mix of upper and lower case	2 bytes	T
%r	Replaced by the time in a.m. and p.m. notation; in the POSIX locale this is equivalent to %I:%M:%S %p	locale, at least 8 bytes	T

Table 5. Conversion Specifiers that Can Be Used in Literal-8 (continued)

Specifier	Description	Length	Allowed For
%R	Replaced by the time in 24 hour notation [%H:%M]	5 bytes	T
%S	Replaced by the second as an integer [00,61]	2 bytes	T
@Sh	Replaced by the hundredths of a second as an integer [00,99]	2 bytes	T
@Sm	Replaced by the millionths of a second as an integer [000000,999999]	6 bytes	T
@So	Replaced by the thousandths of a second as an integer [000,999]	3 bytes	T
@St	Replaced by the tenths of a second as an integer [0,9]	1 bytes	T
%y	Replaced by the year without century as an integer [00,99]	2 bytes	D
%Y	Replaced by the year with century as an integer	usually 4 bytes	D
@Y	Replaced by the year with century as an integer	4 bytes	D
%%	Replaced by a %	1 byte	D, T
@@	Replaced by a @	1 byte	D, T

Table 5 notes:

1. Conversion specifiers are case-sensitive.
2. The Allowed For column symbols have the following meaning:
   - D - DATE item
   - T - TIME item
3. The Length column is based on the default COBOL locale, which is an EBCDIC single-byte encoding scheme (CCSID 37).
4. By default, a value of zero represents the twentieth century (1900 to 1999). This value is based on the base century specified in the DATTIM PROCESS statement option.

[End of IBM Extension]

SIZE Phrase

The SIZE phrase specifies the total size of the date or time item in number of digits. The number of digits must be greater than or equal to the size of the format literal. The size of the format literal is determined by replacing the conversion specifiers with their largest value, and doing conversions, if necessary, to the runtime CCSID. For more information refer to the description of the CCSID parameter for CRTCBLMOD described in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

The SIZE phrase must be specified for a date or time item when the length of that item cannot be determined at compile time. The compiler cannot determine the size of a date or time item when:

- Both literal-8 and the LOCALE phrase are specified, which means the actual length of the date or time item will be partially determined at runtime from the specified locale.
- Literal-8 is specified without a LOCALE phrase, and one of the conversion specifications within literal-8 may result in a variable length item.
- Literal-8 is *not* specified, which means the actual length of the date or time item will be completely determined at runtime from the specified locale.

integer-3, integer-4
Integer-3 and integer-4 specify the size of the default date or time item in number of digits. Integer-3 or integer-4 must be specified if the size of the date or time item cannot be determined at compile time. For a date and time item, integer-3 and integer-4 must be equal to or greater than 4. The maximum size of an item of class date-time is 256, if the item has a USAGE of DISPLAY, or 31 for a USAGE of PACKED-DECIMAL.

LOCALE Phrase

The LOCALE phrase is used to specify the culturally specific locale that is to be used for formatting date and time items.

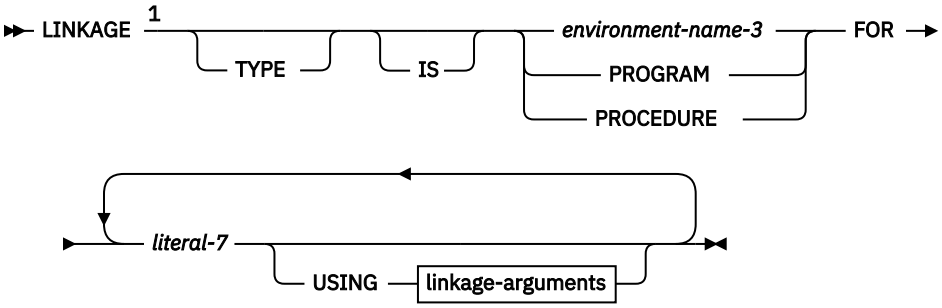
When the LOCALE phrase is specified *without* literal-8, the date or time item's format and separator is completely based on a locale. When the LOCALE phrase is specified *with* literal-8, literal-8 determines the format of the item, but the value used to replace any conversion specifier that is dependent on a locale for its exact representation (for example, %p) will be based on the locale.

mnemonic-name-3, mnemonic-name-4
If mnemonic-name-3 or mnemonic-name-4 *is* specified, the locale used for the date or time item is the one associated with mnemonic-name-3 or mnemonic-name-4 in the SPECIAL-NAMES paragraph. If mnemonic-name-3 or mnemonic-name-4 *is not* specified, the current locale is used. To determine the current locale, refer to the description in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

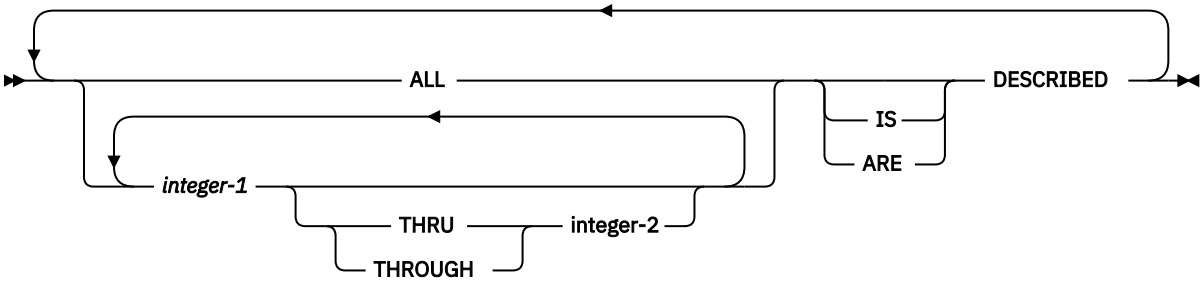
Mnemonic-name-3 and mnemonic-name-4 must be locale mnemonic names. Locale mnemonic names are specified with the LOCALE clause of the SPECIAL-NAMES paragraph see "LOCALE Clause" on page 93.

[IBM Extension] LINKAGE TYPE Clause

The LINKAGE TYPE clause specifies the type of linkage to be made on a CALL to or a CANCEL of the program specified by literal-7, and to the type of linkage to be made on the SET statement.



linkage-arguments



LINKAGE TYPE Clause - Format
Notes:

[1] IBM Extension

environment-name-3
Environment-name-3 can be defined as:

PGM
Linkage to a program object (*PGM) is generated.

PRC
Linkage to an ILE procedure is generated.

SYS
Linkage to a system-supplied procedure is generated.

PROGRAM
Linkage to a program object (*PGM) is generated. This is synonymous to an environment-name-3 of PGM.

PROCEDURE
Linkage to an ILE procedure is generated. This is synonymous to an environment-name-3 of PRC.

literal-7
Literal-7 is the name of the program object or procedure. Literal-7 can contain an extended-name. It can be at most 10 characters long for program names and 256 characters long for procedure names. Literal-7 is affected by the OPTION(*MONOPRC) parameter. When *MONOPROC is specified, lowercase characters are converted to uppercase and the rules for formation of a program-name are followed. See **program-name** in "PROGRAM-ID Paragraph" on page 70 for details.

USING
Specifies which parameters are to have their operational descriptors made available to the called procedure. These parameters must be defined as elementary data items with a USAGE of DISPLAY or DISPLAY-1. They may not be reference modified.

The USING clause is allowed for a linkage type of procedure and applies only to a CALL statement.

integer-1, integer-2
Must be a positive non-zero integer. Specifies the ordinal position of any parameter described using operational descriptors.

Integer-2 must be greater than integer-1.

DESCRIBED
The parameters specified by integer-1 through integer-2 are passed along with corresponding operational descriptors. If ALL is specified, all parameters defined for the procedure are passed along with corresponding operational descriptors, where applicable.

[End of IBM Extension]

LINKAGE TYPE Clause Considerations

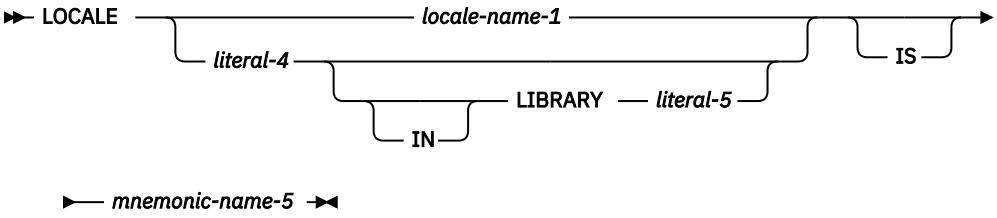
There are several ways to affect the type of linkage generated for a CALL, CANCEL, or SET. They are listed in order of precedence. The LINKAGE phrase of the CALL, CANCEL, or SET statement has the highest precedence. If no LINKAGE phrase is specified on the statement and there is no visible nested program, the LINKAGE TYPE clause is used if specified. The order of precedence is:

- The LINKAGE phrase of the statement
- CALL or CANCEL to a nested program
- The LINKAGE TYPE clause of the SPECIAL-NAMES paragraph
- The LINKLIT parameter of the CRTCBLMOD or CRTBNDCBL command

[IBM Extension] LOCALE Clause

The LOCALE clause is used to define locale mnemonic names and their IBM i equivalent locale object name and library.

LOCALE Clause - Format



locale-name-1
Specifies a system-specific name that refers to a locale object. For ILE COBOL, the only supported locale-name-1 is POSIX.

literal-4
Literal-4 must be a locale object name. It must be a nonnumeric literal with a maximum length of 10 characters.

literal-5
Literal-5 is used to specify the name of the operating system library in which the locale object is to be found. It must be an nonnumeric literal with a maximum length of 10 characters. The special value *LIBL (search using the job's library list) may be specified. If the LIBRARY phrase is omitted , the job's library list is used to search for the locale object.

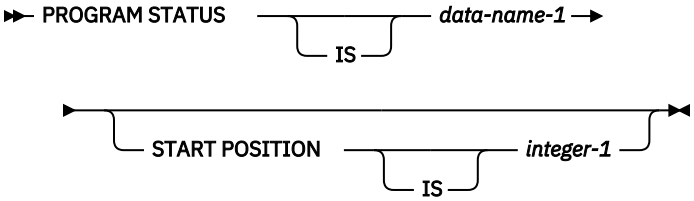
mnemonic-name-5
Mnemonic-name-5 provides a reference to the locale identified by locale-name-1 or the values specified for literal-4 and literal-5. It can only be used in a FORMAT clause, PICTURE clause, Format 8 of the SET statement, or in the argument list of some intrinsic functions.

[End of IBM Extension]

[IBM Extension] PROGRAM STATUS Clause

The PROGRAM STATUS clause specifies a data item into which values from the predefined program status structure are moved after an error occurs in the program.

PROGRAM STATUS Clause - Format



data-name–1
Must be an alphanumeric field described in the WORKING-STORAGE SECTION. If data-name-1 is referenced from a nested program, it must be defined as global in the outermost program. The length of data-name-1 must be in multiples of the lengths of the program status structure subfields.

integer–1
Specifies the start position of the program status structure. If integer-1 is not specified, then the start position is assumed to be 0. Integer-1 must match the start position of a program status structure subfield..

If the PROGRAM STATUS clause is specified in the SPECIAL-NAMES paragraph, data-name-1 is updated with values from the predefined program status structure. This structure contains subfields that provide you with information about the program exception/error that occurred. Table 6 on page 95 provides the layout of the subfields of the data structure and the information that it contains.

Table 6. Contents of the Program Status Data Structure

Start position	Length	Format	Description
0	10	Character	Program name.
10	10	Character	Program library name.
20	10	Character	Module name.
30	10	Character	Statement number. *N if not available.
40	6	Character	Optimization level.
46	7	Character	Exception message identifier.
53	10	Character	Job name.
63	6	Character	Job number.
69	1	Character	Job type.
70	10	Character	User profile running the program.
80	14	Character	Timestamp (in the format YYYYMMDDHHMMSS) for the time that the error occurred.

You select the subfield(s) from the program status structure that gets moved into data-name-1 by coding its length and the start position. The compiler uses the length and start position to determine the program status subfield(s) that data-name-1 gets mapped onto. The length and start position must match one or more predefined subfields of the program status structure.

[End of IBM Extension]

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 specifies information needed for transmission of data between the external medium and the COBOL program.

File Categories

The IBM i 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. A database file is subdivided into groups of records called members. There are two types of database files: physical files and logical files.

A **physical file** is a file that contains data records (similar to disk files on other systems).

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), and can exclude and reorder the fields defined in the physical file(s).

Distributed Files

Distributed files allow a database file to be spread across multiple IBM i servers, while retaining the look and capability of a single database. Performance of large queries can be enhanced by splitting database

requests across multiple systems. Distributed files behave in much the same way as DATABASE files. However, since files are distributed across multiple systems, the arrival sequence or relative number cannot be relied upon, and additional time is required for the data link to pass the data between the systems whenever the remote system is accessed.

For more information about accessing distributed files, refer to the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

Device Files

A device file reads from or writes to a device or remote system. It 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. You can retrieve, add, update, or delete data records in a file that resides on another system.

For more information about accessing remote files, refer to the *Db2® for i* section of the *Database and File Systems* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

Save Files

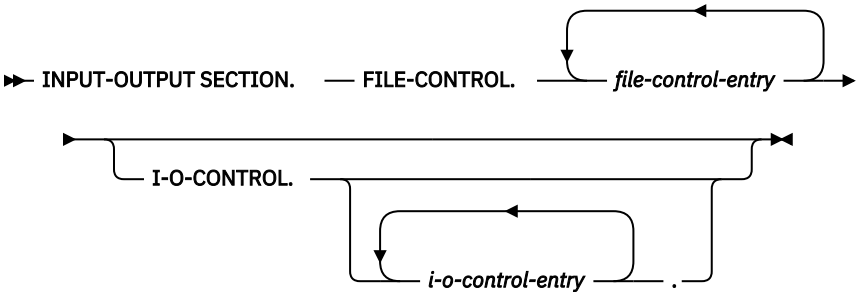
A save file is a file that is used to prepare data in a format that is correct for backup and recovery purposes or for transportation to another system. It contains the output that is produced from the Save Library (SAVLIB) or Save Object (SAVOBJ) CL commands. For information about save files, see the *Db2 for i* section of the *Database and File Systems* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

Paragraphs

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

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

INPUT-OUTPUT Section - Format



FILE-CONTROL paragraph
Names and associates the files with the external media.

The keyword 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 97.

I-O-CONTROL paragraph
Specifies information needed for 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 116.

The exact contents of the Input-Output Section depend on the file organization and access methods used. See “ORGANIZATION Clause” on page 104 and “ACCESS MODE Clause” on page 106.

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.

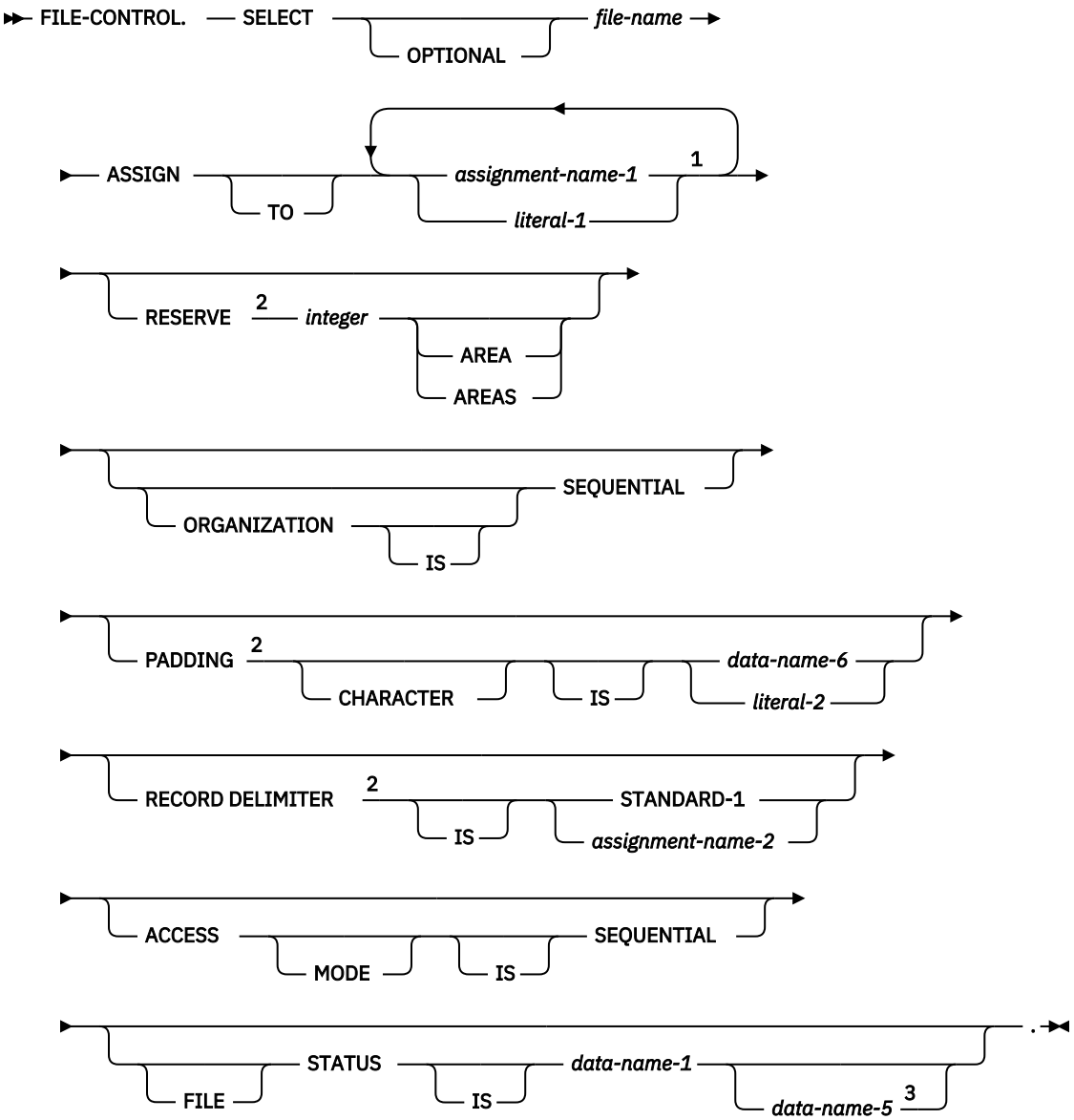
COBOL allows for four distinct kinds of file input and output:

- Sequential
- Relative
- Indexed
- [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.

FILE-CONTROL Paragraph - Format 1 - Sequential Files



FILE-CONTROL Paragraph - Format 1 - Sequential

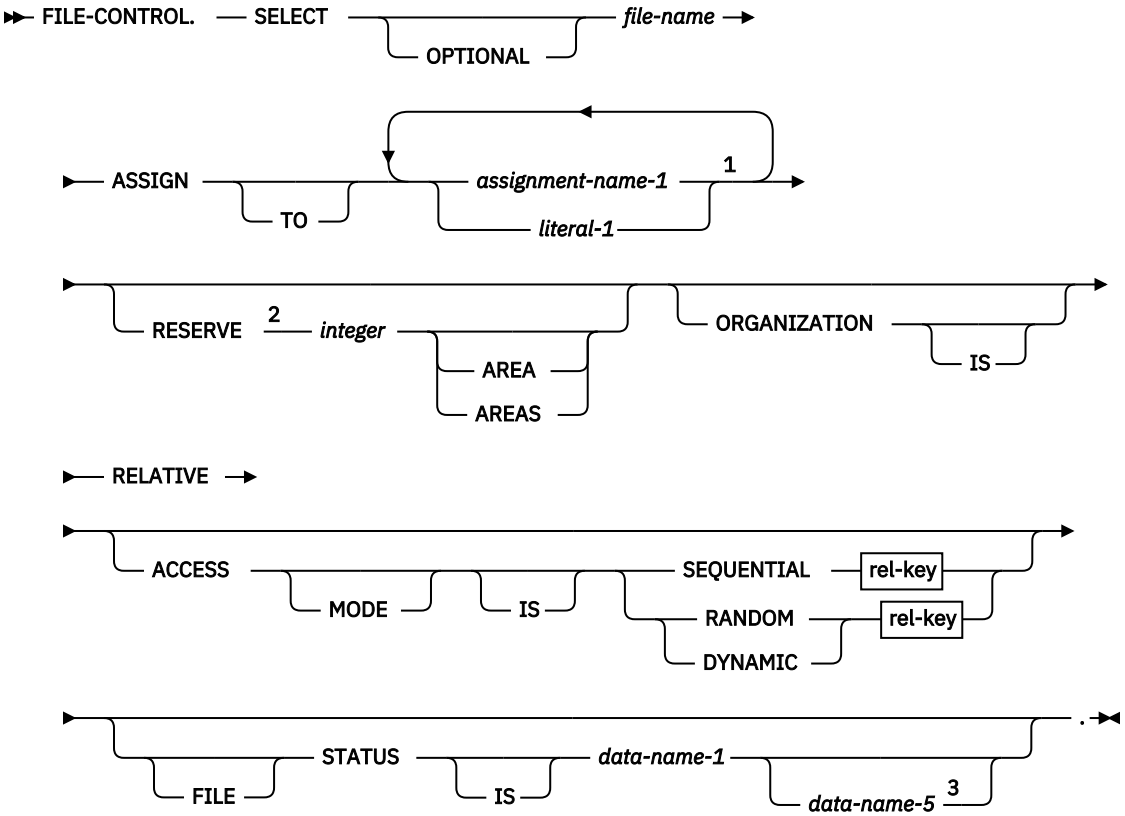
Notes:

[1] Subsequent repetitions syntax-checked only.

[2] Syntax-checked only.

[3] IBM Extension

FILE-CONTROL Paragraph - Format 2 - Relative Files



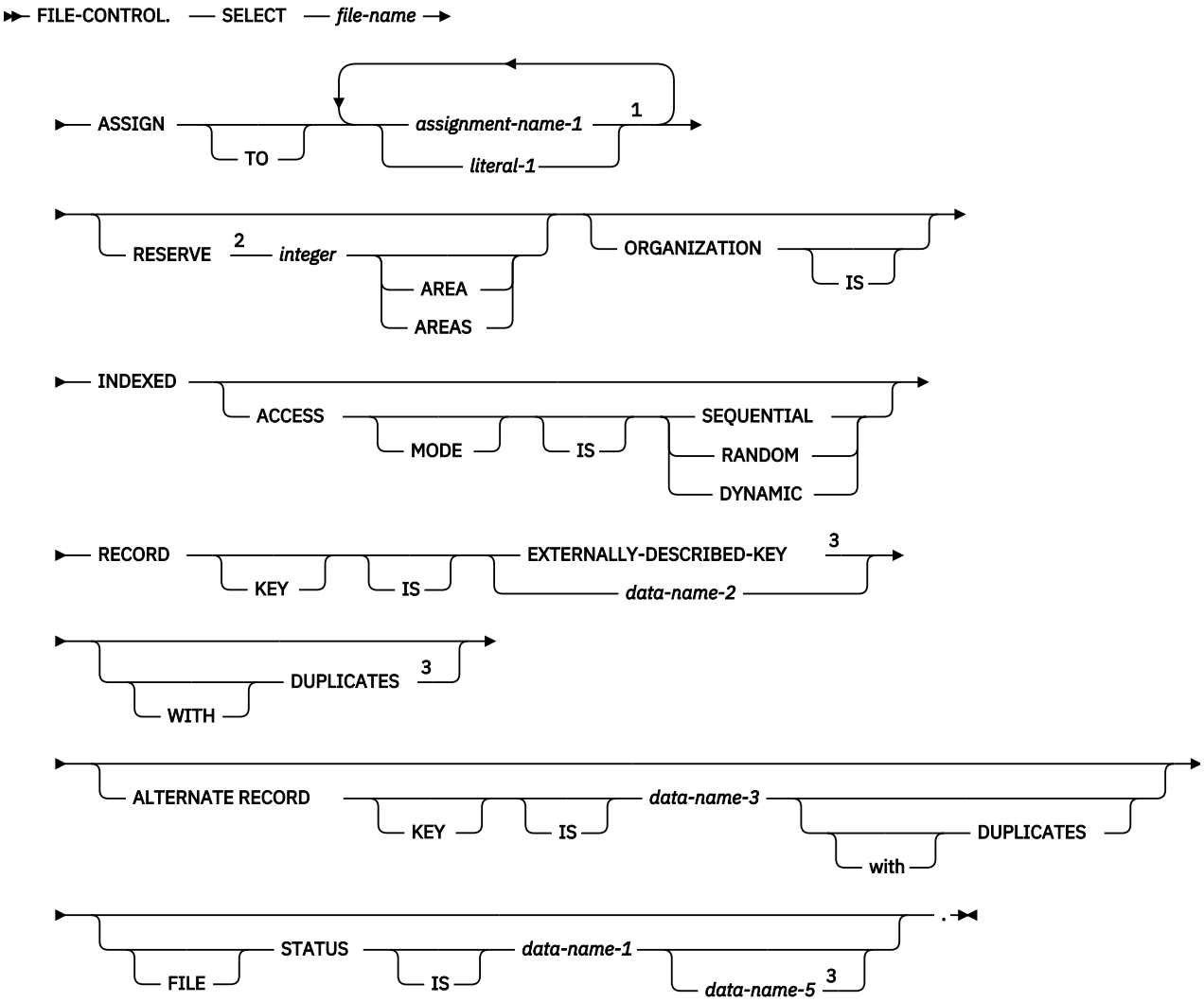
rel-key

FILE-CONTROL Paragraph - Format 2 - Relative

Notes:

[1] Subsequent repetitions syntax-checked only.
[2] Syntax-checked only.
[3] IBM Extension

FILE-CONTROL Paragraph - Format 3 - Indexed Files



FILE-CONTROL Paragraph - Format 3 - Indexed

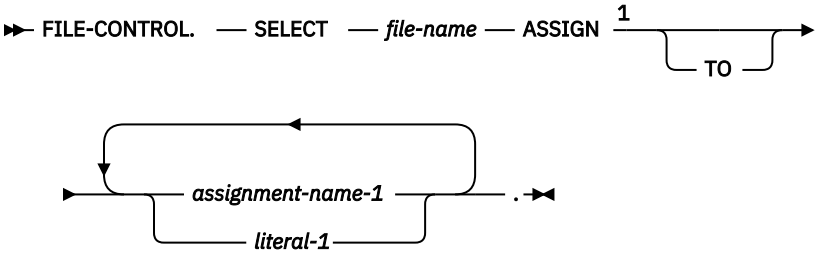
Notes:

[1] Subsequent repetitions syntax-checked only.

[2] Syntax-checked only.

[3] IBM Extension

FILE-CONTROL Paragraph - Format 4 - Sort or Merge Files

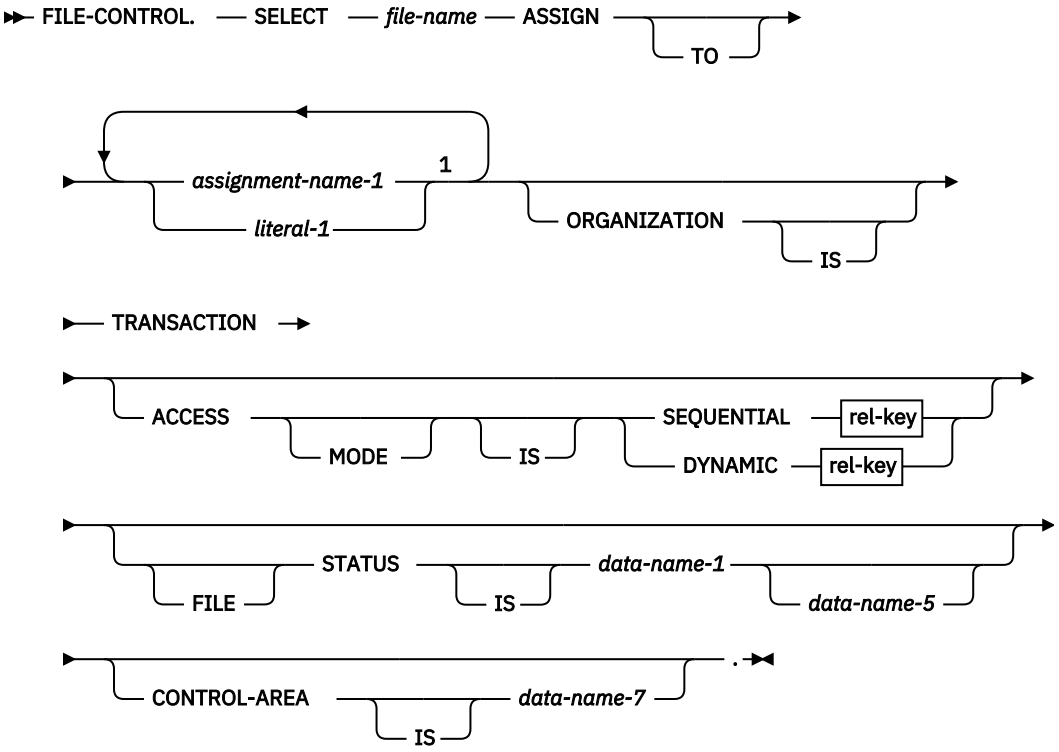


FILE-CONTROL Paragraph - Format 4 - Sort or Merge

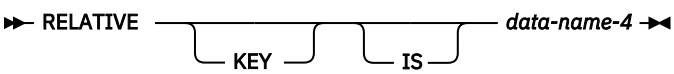
Notes:

[1] Syntax checked only.

[IBM Extension] FILE-CONTROL Paragraph - Format 5 - Transaction Files



rel-key



FILE-CONTROL Paragraph - Format 5 - Transaction

Notes:

[1] Subsequent repetitions syntax checked only.

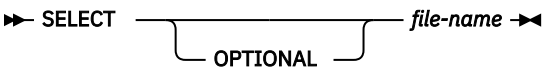
See the chapter on Transaction Files in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information on working with transaction files.

[End of IBM Extension]

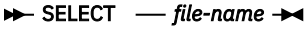
SELECT Clause

The SELECT clause chooses a file.

SELECT Clause - Format - Sequential & Relative Files



SELECT Clause - Indexed, Sort/Merge, & Transaction Files



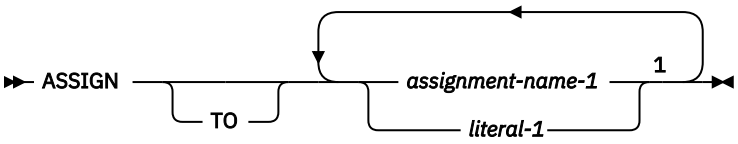
SELECT OPTIONAL (Format Sequential & Relative Files)

May be specified only for sequential and relative 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.

ASSIGN Clause

The ASSIGN clause associates a file with an external medium.



ASSIGN Clause - Format
Notes:

[1] Subsequent repetitions syntax checked only.

For sort or merge files (associated with an SD entry), no external medium is used. The related ASSIGN clause is syntax checked only.

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

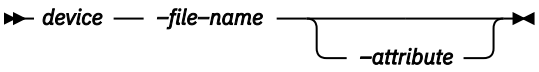
Any assignment-name-1 or literal-1 after the first is syntax checked, but has no effect on the execution of the program

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 *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for further information.

Note:

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. For further information on overriding files, refer to the *File Systems and Management* section of the *Database and File Systems* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

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

FORMATFILE
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 *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

TAPEFILE
Tape file

DISKETTE
Diskette file

DISK
Any physical database file or single format logical database file. When DISK is the device, database extensions cannot be used, but dynamic file creation is supported. See the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information about DISK files. See "OPEN Statement Considerations" on page 348 for information about Dynamic File Creation.

DATABASE
Any database file (or DDM file). When DATABASE is the device, externally described data and database extensions can be used, but dynamic file creation is not supported.

WORKSTATION
Display file or ICF file.

File Name

This part of assignment-name must be a 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.

A quoted file name can be specified within literal-1. For example, if an IBM i system file has a quoted name of "sysfile", the entry for literal-1 is coded as follows:

```
"device-""sysfile""-SI"
```

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 IBM i 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* or *ALWNULL*.

SI
Indicates that a separate indicator area has been specified in the DDS for a FORMATFILE or WORKSTATION file.

ALWNULL
When ALWNULL is specified, the program can manipulate null-capable fields in a database file. This keyword can only be used with device type DATABASE.

See the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for details on the use of the SI or ALWNULL 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 Table 7 on page 104.

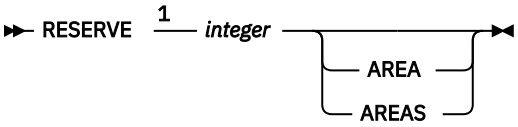
Table 7. Valid Entries for Assignment-Name-1 and Literal-1

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

Key:
R=Required
O=Optional
N=Not Allowed

RESERVE Clause

The RESERVE clause reserves input-output areas. It is syntax checked, but treated as documentation.



RESERVE Clause - Format

Notes:

[1] Syntax-checked only.

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.

[IBM Extension]

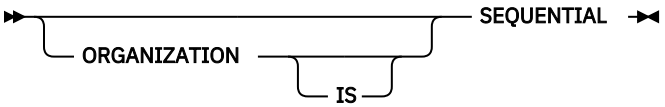
For database files, the ORGANIZATION clause indicates the current program usage of the file in the program. Therefore, the same database file can use SEQUENTIAL, RELATIVE, or INDEXED (assuming a keyed sequence access path exists) in the ORGANIZATION clause. This is true regardless of what is specified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

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)

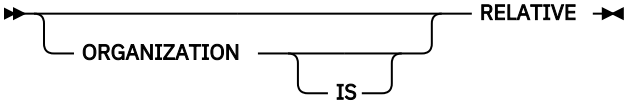
ORGANIZATION Clause - Sequential Files



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 RELATIVE (Format 2)

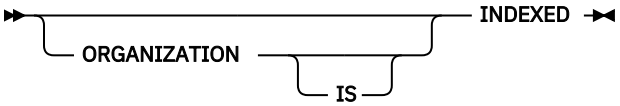
ORGANIZATION Clause - Relative Files



The position of each record in the file is determined by its relative record number within the arrival sequence access path.

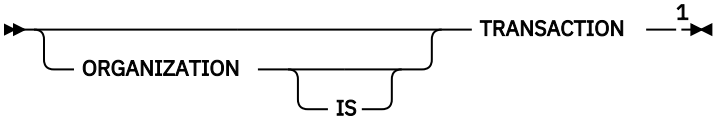
ORGANIZATION IS INDEXED (Format 3)

ORGANIZATION Clause - Indexed Files



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.

[IBM Extension] ORGANIZATION IS TRANSACTION (Format 4)



ORGANIZATION Clause - Transaction Files

Notes:

[1] IBM Extension

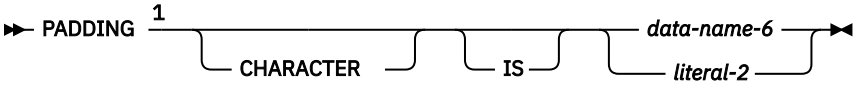
Signifies interaction between a COBOL program and either a workstation user or another system. For more information on transaction files, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

[End of IBM Extension]

PADDING CHARACTER Clause

The PADDING CHARACTER clause specifies the character which is to be used for block padding on sequential files.

The PADDING CHARACTER clause is syntax checked, but no compile-time or run-time verification checking is done, and the clause has no effect on the execution of the program.



PADDING CHARACTER Clause - Format

Notes:

[1] Syntax-checked only.

data-name-6
Must be defined in the Data Division as an alphanumeric one-character data item, and must not be defined in the File Section. Data-name-6 can be qualified.

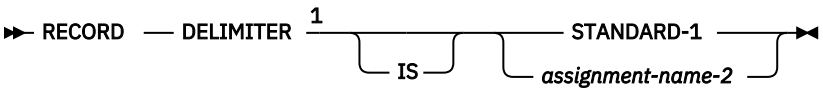
literal-2
Must be a 1-character nonnumeric literal.

For EXTERNAL files, if data-name-6 is specified, it must reference an EXTERNAL data item.

RECORD DELIMITER Clause

The RECORD DELIMITER clause indicates the method of determining the length of a variable-length record on an external medium. It can be specified only for variable-length records.

The RECORD DELIMITER clause is syntax checked, but no compile-time or run-time verification checking is done, and the clause is treated as documentation



RECORD DELIMITER Clause - Format

Notes:

[1] Syntax-checked only.

STANDARD-1
If STANDARD-1 is specified, the external medium must be a magnetic tape file.

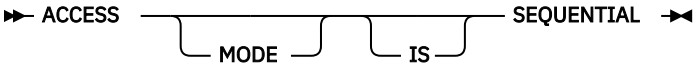
assignment-name-2
Can be any COBOL word.

ACCESS MODE Clause

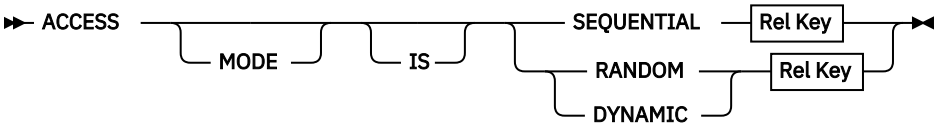
The ACCESS MODE clause defines the manner in which the records of the file are made available for processing. If the ACCESS MODE clause is not specified, SEQUENTIAL access is assumed.

ACCESS MODE Clause - Format 1 - Sequential Files

ACCESS MODE Clause - Format 1 - Sequential Files

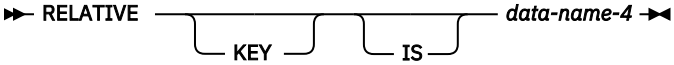


ACCESS MODE Clause - Format 2 - Relative Files



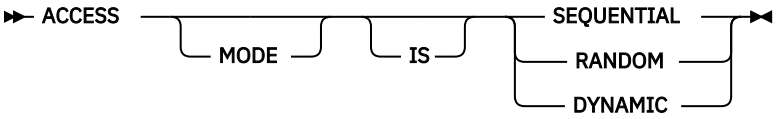
ACCESS MODE Clause - Format 2 - Relative Files

Rel Key



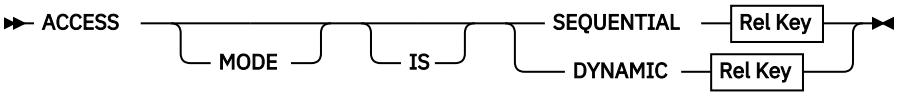
ACCESS MODE Clause - Format 3 - Indexed Files

ACCESS MODE Clause - Format 3 - Indexed Files



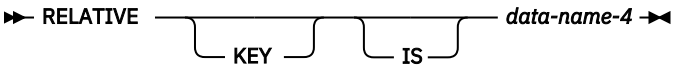
ACCESS MODE Clause - Format 4 - Transaction Files

[IBM Extension]



ACCESS MODE Clause - Format 4 - Transaction Files

Rel Key



[End of IBM Extension]

ACCESS MODE IS SEQUENTIAL

Can be specified for all three kinds of files.

Sequential

Records in the file are accessed in the sequence established when the file was created or extended (arrival sequence).

Relative

Records in the file are accessed in the ascending sequence of relative record numbers of existing records in the file.

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]

ACCESS MODE IS RANDOM

Can be specified for relative and indexed files only. Also, ACCESS MODE IS RANDOM must not be specified for file names specified in the USING or GIVING phrase of a SORT or MERGE statement.

Relative

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

Indexed

The value placed in a record key data item for the current key of reference specifies the record to be accessed.

ACCESS MODE IS DYNAMIC

Can be specified for relative and indexed files only.

Relative

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

Indexed

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. 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
- Relative
- Indexed
- [IBM Extension] Transaction [End of IBM Extension]

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.

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. Relative files must be assigned to DISK or DATABASE.

Table 8 on page 108 summarizes conditions affecting relative output files.

Table 8. 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 recover from a file status of 9Q, use the CHGPF (Change Physical File) 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.

Extending the file boundary

After file creation time, the size of a file can be extended. If a file status 0Q is received for a file, you may need to add more records to the file before processing it. You can use the INZPFM (Initialize Physical File Member) command to add deleted records to the file.

For example, suppose you create a file of 10 000 records with 3 increments of 1 000 records each:

1. You initialize the (first 10 000) records.
2. You realize you need to store more data. So, you run the INZPFM command with the RECORDS(*DLT) option again, until you have all 13 000 records initialized.
3. You receive a requirement to store even more data - but you have already used up all 13 000 records! If you run the INZPFM command again, you will receive an interactive error message (of severity 99) prompting you either to
   a. Cancel the INZPFM request
   b. Go ahead with the request (say, initialize another 1 000 records).
4. If you choose the second option in the previous step, you now have 14 000 initialized records. You have thus increased the size of the file past the previously defined maximum.

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 and DISK files can have indexed organization.

When records are inserted, updated, or deleted, they are identified solely by the values of their prime keys. Specify 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]

[IBM Extension] TRANSACTION Organization

Workstation and data communication files can have TRANSACTION organization. See the Transaction Files chapter in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

[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 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 are accessed sequentially. The sequence in which records are accessed is the order in which the records were originally written.

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 from sequential access to random access, using the appropriate forms of input-output statements.

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. Records having the same duplicate value in an alternate record key which is the key of reference are made available in the same order in which they were released by execution of WRITE statements, or REWRITE statements which create such duplicate values.

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 the RECORD KEY data item. If a set of records has alternate record key values, only the first record written is available.

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

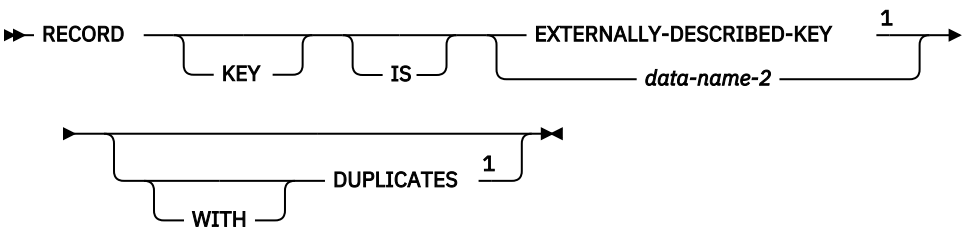
[IBM Extension] Transaction Files

See the Transaction Files chapter in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for a discussion of access mode considerations for transaction files.

[End of IBM Extension]

RECORD KEY Clause

The RECORD KEY clause 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.



RECORD KEY Clause - Format

Notes:

[1] IBM Extension

DUPLICATES Phrase

[IBM Extension]

The DUPLICATES phrase can only be specified for files assigned duplicate record keys. 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. It must not reference a group item that contains a variable occurrence data item. Data-name-2 may be qualified, but it must not be subscripted.

The length of the record key is restricted; the key length, in bytes, cannot exceed 2 000. For more information, see the *Db2 for i* section of the *Database and File Systems* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

If the indexed file contains variable-length records, data-name-2 must be contained within the first "x" positions of the record, where "x" equals the minimum record size specified for the file.

For EXTERNAL files, all file description entries in the run unit that are associated with the EXTERNAL file must specify the same data description entry for data-name-2 with the same relative location within the associated record; otherwise the results are undefined.

[IBM Extension] The RECORD KEY data item, data-name-2, can be a date-time item or numeric item when the file is assigned to a DATABASE device type. The numeric item can have a usage of DISPLAY, COMP-1, COMP-2, COMP (COMP-3), COMP-4, COMP-5, PACKED-DECIMAL, or BINARY. The numeric item can also be an external floating-point data item. [End of IBM Extension]

[IBM Extension] ILE COBOL supports a wide range of date and time data item formats. Many of these formats are not supported by DDS; in this case, the underlying DDS field must be defined as a character or numeric field. In cases where COBOL defines a date-time item, but the underlying DDS field is not date-time, retrieving or writing records to the database will be in the order determined by the underlying DDS data type. [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] EXTERNALLY-DESCRIBED-KEY

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.

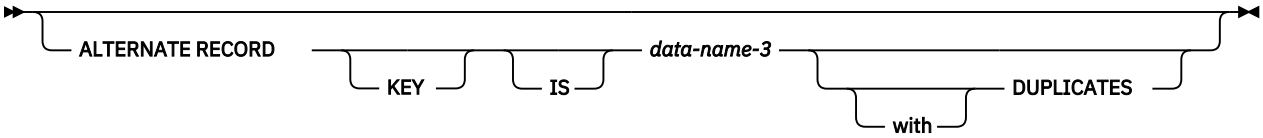
If a portion of the key is declared in the logical file only as an element of a concatenated item (rather than an independently-declared item), the result of the CONCAT operation must not be a variable-length item.

The reserved word EXTERNALLY-DESCRIBED-KEY cannot be specified with the ALTERNATE RECORD KEY clause.

[End of IBM Extension]

ALTERNATE RECORD KEY

The ALTERNATE RECORD KEY clause specifies the data item within the record that is an alternate record key for an indexed file. These alternate keys allow the ILE COBOL program to access the file using a different logical ordering of the file records.



data-name-3

Data-name-3 is the ALTERNATE RECORD KEY data item. It must be described as a fixed length alphanumeric item within a record description entry associated with the file. It must not reference a group item that contains a variable occurrence data item. Data-name-3 may be qualified, but it must not be subscripted.

The length of the alternate record key is restricted; the alternate key length, in bytes, cannot exceed 2000. If the indexed file contains variable-length records, data-name-3 must be contained within the first "x" positions of the record, where "x" equals the minimum record size specified for the file.

For EXTERNAL files, all file description entries in the run unit that are associated with the EXTERNAL file must specify:

- the same data description entry for data-name-3
- the same relative location within the associated record
- the same number of alternate record keys
- the same DUPLICATES phrase.

[IBM Extension] The ALTERNATE RECORD KEY data item, data-name-3, can be a date-time item or numeric item when the file is assigned to a DATABASE device type. The numeric item can have a usage of DISPLAY, COMP-1, COMP-2, COMP (COMP-3), COMP-4, COMP-5, PACKED-DECIMAL, or BINARY. The numeric item can also be an external floating-point data item. ILE COBOL supports a wide range of date and time data item formats. Many of these formats are not supported by DDS; in this case, the underlying DDS field must be defined as a character or numeric field. In cases where ILE COBOL defines a date-time item, but the underlying DDS field is not a date-time, retrieving records will be in the order determined by the underlying DDS data type. [End of IBM Extension]

[IBM Extension] The keys are ordered within the collating sequence used when the file was created. The data description of data-name-3, its relative location within the record and its length must be the same as those used when the file was defined in DDS. The leftmost character position of data-name-3 must not be the same as the leftmost character position of the RECORD KEY or of any other ALTERNATE RECORD KEY. If the DUPLICATES phrase is not specified, the values contained in the ALTERNATE RECORD KEY data item must be unique among records in the file. If the alternate key index is temporary, the order of retrieval of duplicate records is not guaranteed to be in any specific order. If the alternate key index is permanent, the DDS file level keywords, LIFO, FIFO, FCFO can be used to specify the order of retrieval of duplicate records. For more information on alternate key indexes, refer to the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*. [End of IBM Extension]

Usage Considerations

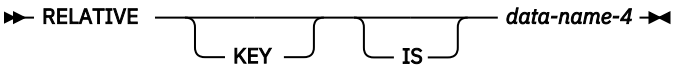
- The sequencing of alternate keys is the same as the primary key. If the primary key spans multiple DDS key fields in the file, the alternate key sequence is determined by the first primary key field. If permanent alternate indexes are used, the key sequence of the logical file must also be the same as the physical file. That is, if the DDS keyword DESCEND is specified in the physical file DDS, it must also be specified in the logical file DDS. Otherwise, ILE COBOL will not be able to find the permanent alternate index.
- Files with alternate keys cannot have a primary record key that is externally defined.
- The maximum number of alternate keys allowed per file is 253.
- Blocking is implicitly disabled for files with alternate keys.
- Parameter values specified on an override command, other than TOFILE, MBR, LVLCHK, WAITRCD, SEQONLY, and INHWRT are ignored when ILE COBOL builds an alternate index.
- In order to use alternate record keys, the database file must meet the following requirements. Otherwise, the OPEN operation will fail and the file status will be set to 39.
  1. The field(s) in the database file that is to be used as an alternate key must be an input, output, or both input/output field.
  2. The database file cannot be a Distributed Data Management (DDM) file.
  3. The database file must not share an open data path.
  4. The DUPLICATES clause specified for each key in the program must match the duplicates attribute of the database file. This includes the primary key. If you are using permanent alternate indexes, the DDS keyword UNIQUE is use to specify unique keys. The absence of this keyword implies that the file allows duplicate keys. If you are using temporary alternate indexes and the DUPLICATES clause is not specified, you must ensure that existing records in the database file do not have duplicate values in the fields that are defined as keys in the program.
- The following will cause an OPEN operation to fail with the file status set to 90.
  1. ILE COBOL will open one additional file for each alternate key. These files are opened with open identifiers that begin with "QARK". Open identifiers must be unique within the activation group that the program is running in. The OPEN operation will fail if ILE COBOL detects a non-unique open identifier. This may be possible if you use the OPNDBF and/or OPNQRYF commands along with your ILE COBOL program and specify open identifiers that begin with "QARK".
  2. If the CRTARKIDX option is not specified, and a permanent index cannot be found by ILE COBOL, the OPEN operation will fail.

3. The maximum number of contiguous DDS fields that can be used to form an alternate key is 156. If this limit is exceeded, the OPEN operation will fail.

RELATIVE KEY Clause

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

RELATIVE KEY Clause - 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 can be qualified.

For reads under sequential access, the RELATIVE KEY data item is updated with the relative record number of the record being made available.

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. For further information, see the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/. [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.

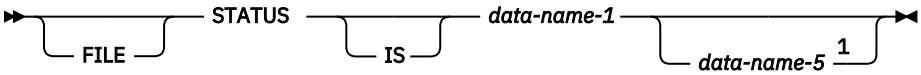
The ACCESS MODE IS RANDOM clause must not be specified for file-names specified in the USING or GIVING phrase of a SORT or MERGE statement.

For EXTERNAL files, data-name-4 must reference an external data item, and the RELATIVE KEY phrase in each associated file control entry must reference that same external data item. (Relative keys are used with subfiles.)

Refer to the Transaction File chapter in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for transaction file considerations.

FILE STATUS Clause

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



FILE STATUS Clause - Format

Notes:

[1] IBM Extension

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 250.)

When the compiler generates code to block output records or unblock input records, file status values that are caused by operating system exceptions are set only when a block is processed. See “Appendix F. File Structure Support Summary and Status Key Values” on page 563 for a description of the possible values. See the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information on blocking output records and unblocking input records.

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. Data-name-1 can be qualified.

[IBM Extension]

data-name-5
An optional status key data item may be specified for file processing.

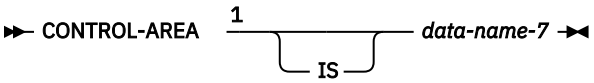
For transaction files, the data item must be a 4-character alphanumeric item.

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. Extended file status is set to 0900 for files that are created dynamically when OPEN OUTPUT is specified. Data-name-5 can be qualified.

[End of IBM Extension]

[IBM Extension] CONTROL-AREA Clause

This clause specifies device-dependent and system-dependent information used to control input/output operations for TRANSACTION files.



CONTROL-AREA Clause - Format
Notes:

[1] IBM Extension

data-name-7
A data-item (2, 12, or 22 characters long) defined in the LINKAGE, LOCAL-STORAGE or WORKING-STORAGE SECTIONS, of the following format:

```
01    data-name-7
      05   function-key  PIC X(2)
      05   device-name   PIC X(10)
      05   record-format PIC X(10)
```

Where:

function-key
Is a 2-digit number inserted in the field by the workstation interface that identifies the function key the operator pressed to initiate the transaction.

Number	Meaning
00	Enter key
01-24	Function keys 1 through 24
90	Roll up / Page down key
91	Roll down / Page up key

92
Print key

93
Help key

94
Clear key

95
Home key

99
Undefined

device-name
The program device name

record-format
The DDS record format name that was referenced by the last input/output statement run.

[End of IBM Extension]

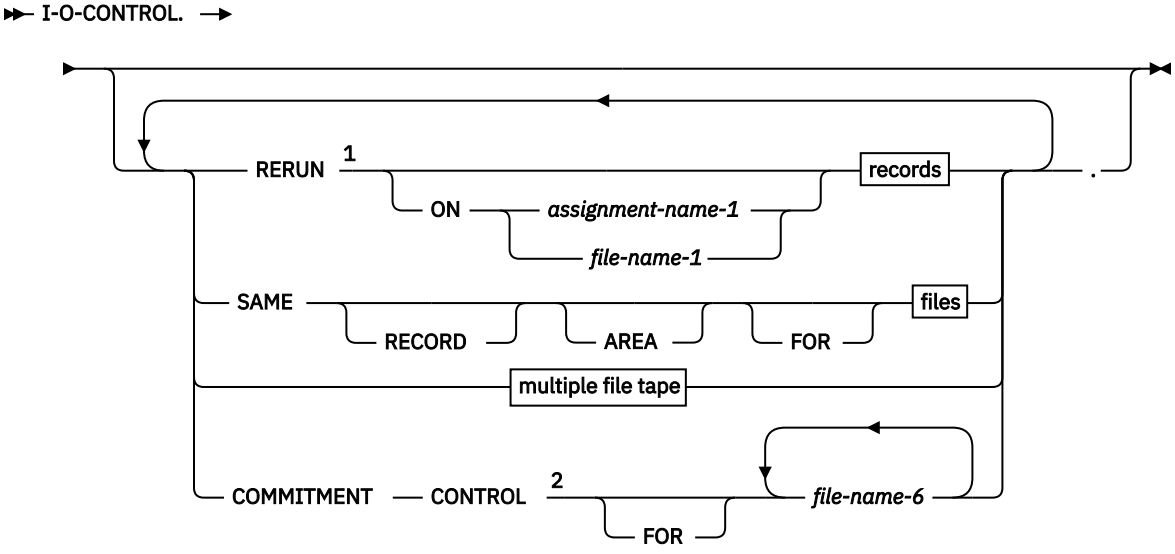
I-O-CONTROL Paragraph

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

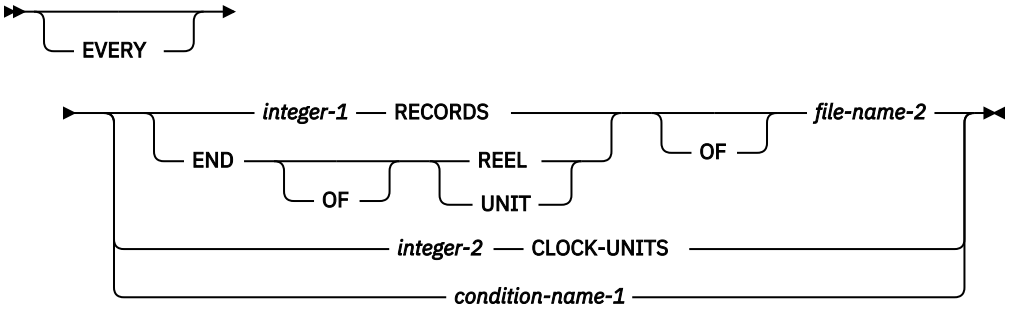
The keyword I-O-CONTROL may appear only once, at the beginning of the paragraph. The word I-O-CONTROL must begin in Area A, and must be followed by a separator period.

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.

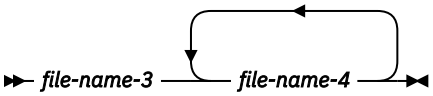
I-O-CONTROL Paragraph - Format 1 - Sequential Files



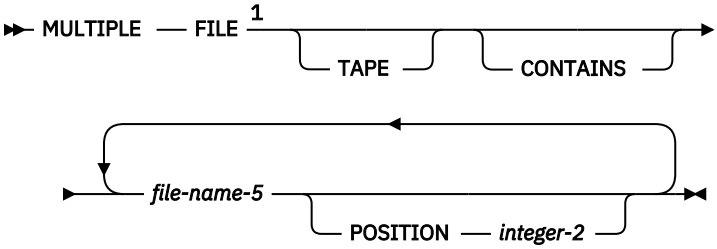
records



files



multiple file tape

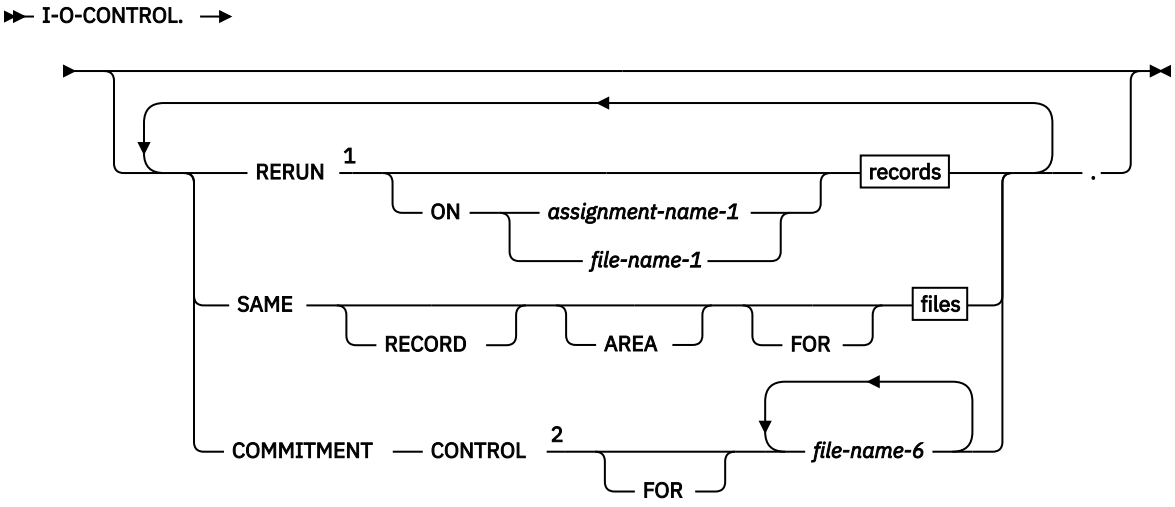


I-O-CONTROL Paragraph - Format 1 - Sequential

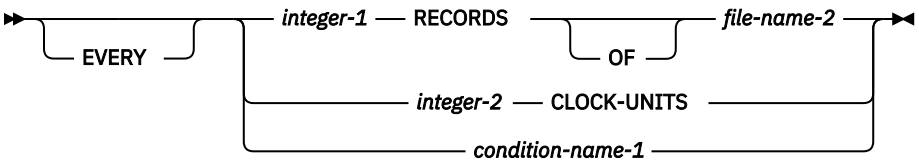
Notes:

[1] Syntax-checked only.
[2] IBM Extension

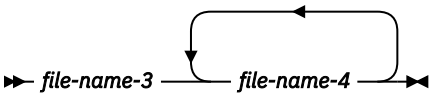
I-O-CONTROL Paragraph - Format 2 - Relative and Indexed Files



records



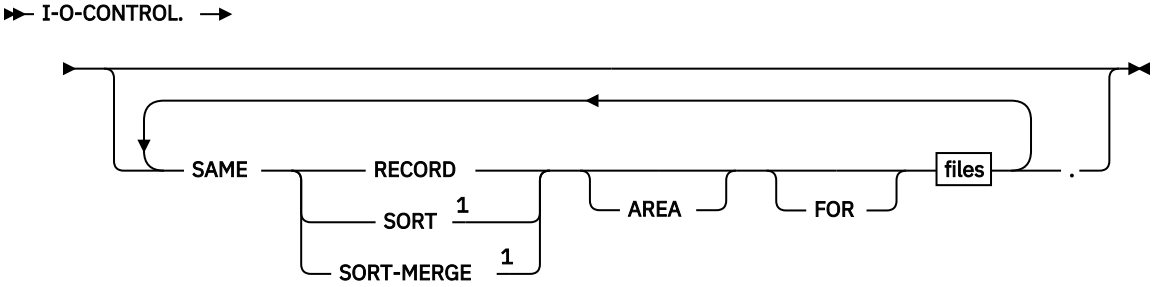
files



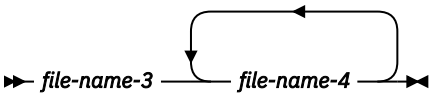
I-O-CONTROL Paragraph - Format 2 - Relative/Indexed
Notes:

[1] Syntax-checked only.
[2] IBM Extension

I-O-CONTROL Paragraph - Format 3 - Sort or Merge Files



files



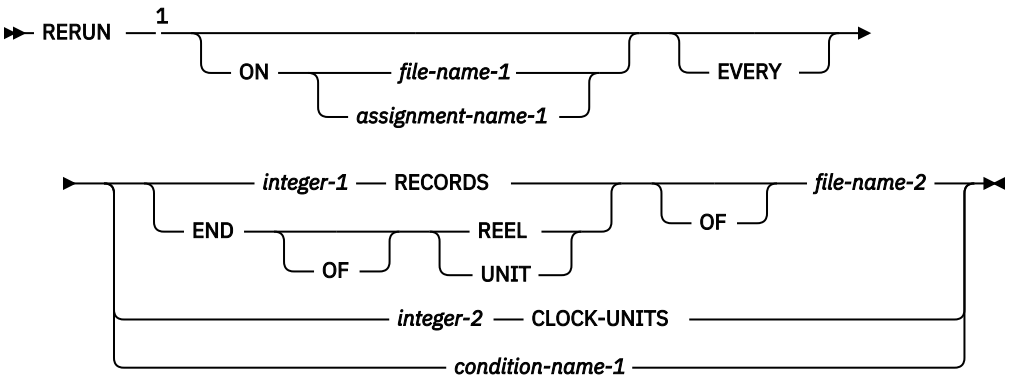
I-O-CONTROL Paragraph - Format 3 - Sort/Merge
Notes:

[1] Syntax-checked only.

RERUN Clause

The RERUN clause specifies that checkpoint records are to be taken.

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



RERUN Clause - Format
Notes:

[1] Syntax-checked only.

file-name-1
The name of a sequentially organized file. The file named in the RERUN clause must be a file defined in the same program as the I-O-CONTROL paragraph, even if the file is defined as GLOBAL.

assignment-name-1
This name can be any user defined word. The file named in the RERUN clause must be a file defined in the same program as the I-O-CONTROL paragraph, even if the file is defined as GLOBAL.

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.

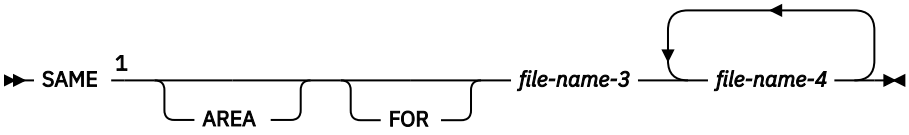
Integer-1 must be an unsigned integer. It specifies the number of records to be processed before the RERUN information is written.

EVERY END OF REEL/UNIT
No two multiple END OF REEL or END OF UNIT phrases can specify the same file-name-2. The definition of UNIT is determined by each assignment-name-1.

EVERY integer-2 CLOCK-UNITS
Only one RERUN clause containing the CLOCK-UNITS phrase can be specified.

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



SAME AREA Clause - Format
Notes:

[1] Syntax-checked only.

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

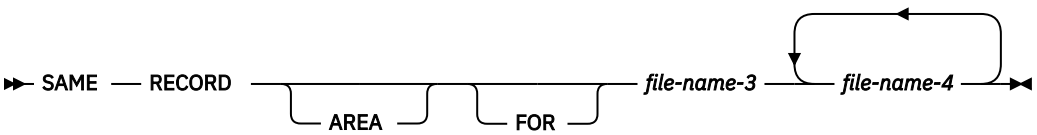
file-name-3, file-name-4, ...
Must be specified in the FILE-CONTROL paragraph of the same program as the I-O-CONTROL paragraph. They cannot reference an external file connector.

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.

Note: The SAME RECORD AREA clause is intended to make efficient use of main storage. However, IBM i 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 clause actually degrades performance and increases program size.

SAME RECORD AREA Clause - Format



A logical record in the shared storage area is considered to be both of the following:

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

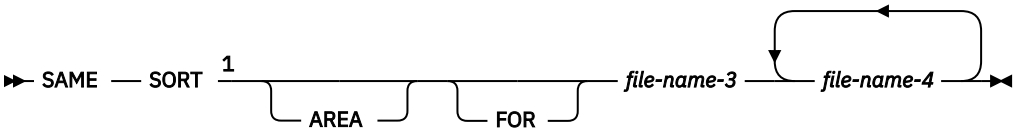
More than one SAME RECORD AREA clause may be included in a program. However:

- 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.
- If the SAME RECORD AREA is specified for several files, the record description entries or the file description entries for these files must not include the GLOBAL clause.
- The SAME RECORD AREA clause cannot be used with EXTERNAL files.

SAME SORT AREA Clause

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

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



SAME SORT AREA Clause - Format

Notes:

[1] Syntax-checked only.

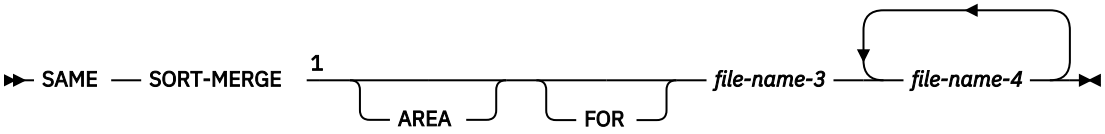
When the SAME SORT AREA clause is specified, at least one file-name specified must be a sort file. Files that are not sort files may also be specified. The following rules apply:

- 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 RECORD AREA clause.

SAME SORT-MERGE AREA Clause

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

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



SAME SORT-MERGE AREA Clause - Format

Notes:

[1] Syntax-checked only.

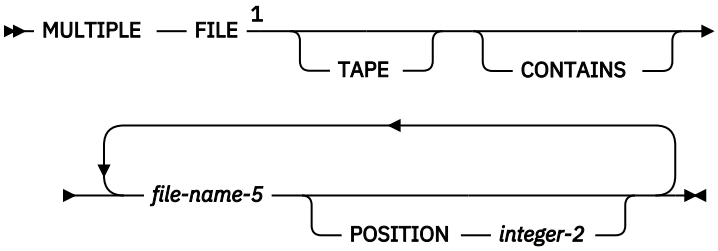
When the SAME SORT-MERGE AREA clause is specified, at least one file-name specified must be a sort or merge file. Files that are not sort or merge files may also be specified. The following rules apply:

- 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 RECORD AREA clause.

MULTIPLE FILE TAPE Clause

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.

The MULTIPLE FILE TAPE clause is syntax checked, but is treated as documentation. See the CRTTAPF, CHGTAPF, and OVRTAPF commands in the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.



MULTIPLE FILE TAPE Clause - Format

Notes:

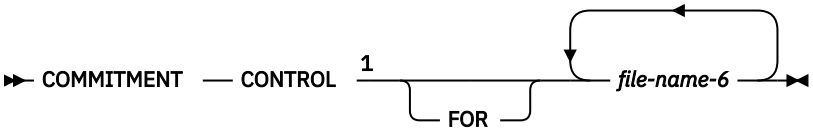
[1] Syntax-checked only.

integer-2
Must be an unsigned integer. It specifies the relative position of the file on the tape.

file-name-5
Names the files that share the tape.

[IBM Extension] COMMITMENT CONTROL Clause

The COMMITMENT CONTROL clause specifies the files that will be placed under commitment control when they are opened.



COMMITMENT CONTROL Clause - Format

Notes:

[1] IBM Extension

File-name-6 must be specified in the FILE CONTROL paragraph of the same program as the I-O-CONTROL paragraph in which the COMMITMENT CONTROL clause appears.

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 *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

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]

Chapter 6. Data Division

Data Division Overview

The Data Division of a COBOL source program describes, in a structured manner, all the data to be processed by the object program; also the relationship between physical and logical records. 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.

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 four sections:

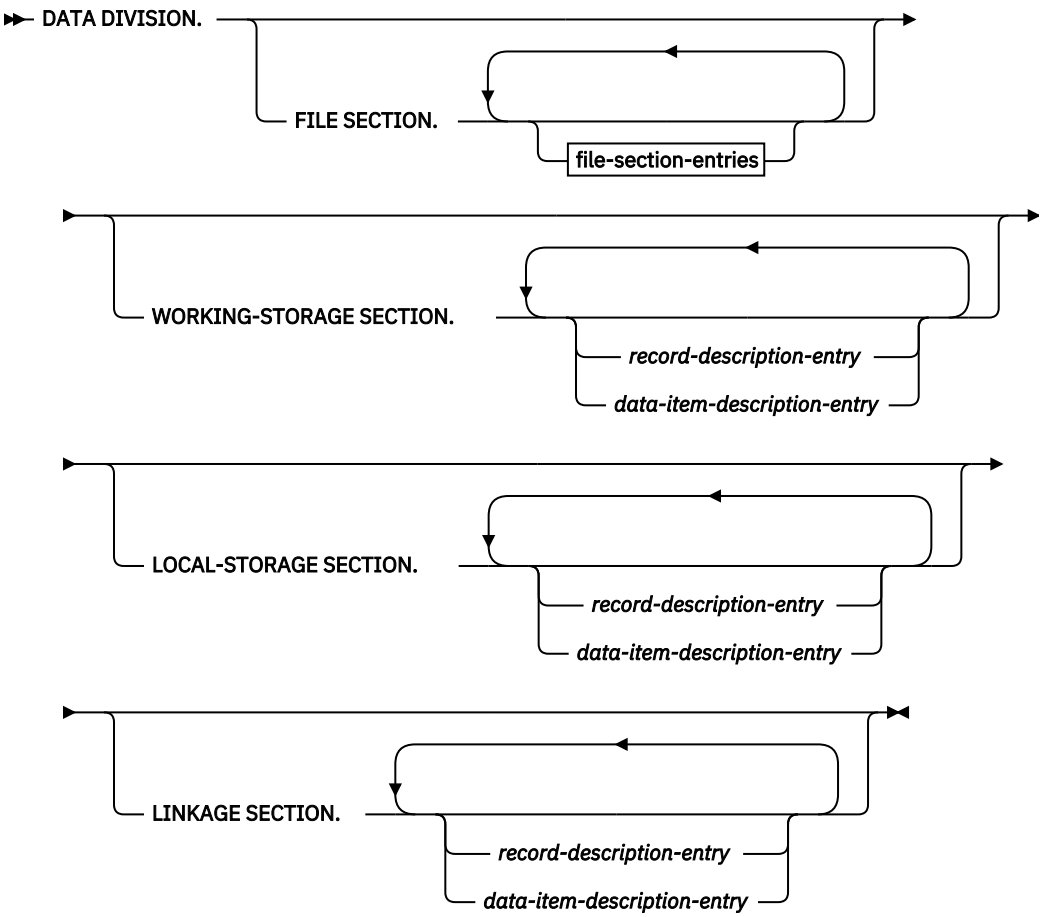
File Section
Describes externally stored data (including sort-merge files).

Working-Storage Section
Describes internal data.

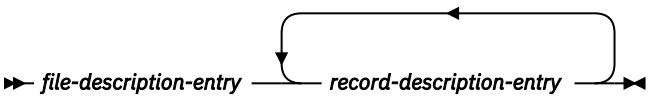
Local-Storage Section
Describes internal data that is allocated on a per-invocation basis.

Linkage Section
Describes data made available by another program. It appears in the called program and describes data items that are provided by the calling program and are referred to by the called program. The called program can be a nested program. For more information on nested programs, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

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.



Data Division - Format
file-section-entries



File Section

The File Section describes:

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

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 133.

record-description-entry
A set of data description entries that describe the particular record(s) contained within a particular file, or describe a type-name (by using the TYPEDEF clause). For the format and the clauses required in a record description entry, see "Data Division—Data Description Entry" on page 150.

More than one record description entry may be specified; each entry that does not describe a type-name 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. Type-names defined in the FILE SECTION may be used in the WORKING-STORAGE, LOCAL-STORAGE, or LINKAGE SECTIONS to define other data items.

Group items (including tables) are limited to a length of 16 711 568 bytes.

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 Chapter 8, "Compiler-Directing Statements," on page 507 for further information on this format of the COPY statement. [End of IBM Extension]

Working-Storage Section

The Working-Storage Section describes data records that are not part of external data files but are developed and processed internally by the program. Type-names may be defined in the WORKING-STORAGE SECTION.

record-description-entry
See "File Section" on page 124 for a 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. Each is defined in a separate data-item description entry that begins with either the level number 77 or 01. For the format and the clauses required in a data-item description entry, see "Data Division—Data Description Entry" on page 150.

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 16 711 568 bytes is permitted for group items (including tables).

[IBM Extension] Local-Storage Section

The Local-Storage Section defines storage that is allocated and freed on a per-invocation basis. On each invocation, data items defined in the Local-Storage Section are reallocated and initialized to the value assigned in their VALUE clauses. Data items defined in the Local-Storage Section cannot specify the EXTERNAL clause. The Local-Storage Section must begin with the header LOCAL-STORAGE SECTION followed by a separator period.

record-description-entry
See "File Section" on page 124 for a description.

data-item-description-entry
See "Working-Storage Section" on page 125 for a description.

You can specify the Local-Storage Section in recursive programs, and in non-recursive programs.

[End of IBM Extension]

Linkage Section

The Linkage Section describes data made available from another program through the CALL statement. It can also be used to describe the format of data accessed by using the ADDRESS OF special register. For example, you can set the ADDRESS OF special register for a Linkage Section item to data that is dynamically allocated using ILE bindable APIs.

record-description-entry
See "File Section" on page 124 for a description.

data-item-description-entry
See "Working-Storage Section" on page 125 for a description.

Record description entries and data item description entries in the Linkage Section provide names and descriptions of the data item, but not the storage. Storage is not reserved in the program because the data area exists elsewhere. Type-names may be defined in the LINKAGE SECTION.

Any data description clause may be used to describe items in the Linkage Section, with these exceptions:

- The VALUE clause may not be specified for items other than level-88 items.

  [IBM Extension] If the VALUE clause is specified for items other than level-88 in the Linkage section, it is treated as a comment. [End of IBM Extension]
- The EXTERNAL clause cannot be specified in the Linkage section.
- The GLOBAL clause cannot be specified in the Linkage section.
- [IBM Extension] The GLOBAL clause can be specified for a data-name or condition-name in the LINKAGE section, with level number 01. When GLOBAL is specified in a LINKAGE section data item, a contained source program can refer directly to the item by the name of the data item. For more information on coding the LINKAGE section, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*. [End of IBM Extension]

Note: A maximum of 16 711 568 bytes is 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. You may take the ADDRESS OF any Data Division item that is not a type-name or a subordinate of a type-name, and that does not have level number 66 or 88. Such an address can be referenced, but not changed.

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

[End of IBM Extension]

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 the ADDRESS OF identifier, it is no longer the starting address of a data structure. It is a calculated address.

You can specify the ADDRESS OF or ADDRESS OF special register as an argument to the LENGTH function. If the ADDRESS OF or ADDRESS OF special register is used as the argument to the LENGTH function, the result is always 16, independent of the identifier specified for ADDRESS OF.

A function-identifier is not allowed in either the ADDRESS OF or the ADDRESS OF special register.

A date-time data item can be used in expressions involving the ADDRESS OF or ADDRESS OF special register.

Types of Data

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

File Data

A **file** is a collection of data records existing on some input-output device. (See “File Section” on page 140.) A file can be considered to be a group of physical records; it can also be considered to be a group of 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).)

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.

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

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 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 156 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. Type-names (defined using the TYPEDEF clause) must be level-01 items.

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.

If a type-name is a group item, and it is used in a TYPE clause to define a new data item, then the new data item will have subordinate items of the same name, description, and hierarchy as those of the type-name. There is no limit to the number of levels that can result because:

- The subject of a TYPE clause may have a level number as high as 49, and a type-name may describe a group item with as many levels as 49
- Type declarations may reference other type declarations.

[IBM Extension] Coding Example

The ILE COBOL 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).
```

Because 04 is less than 05, it is not subordinate to EMPLOYEE-NAME, yet because it is greater than 01 it is subordinate to EMPLOYEE-RECORD. If 07 was used in place of 04, EMPLOYEE-ADDRESS would be subordinate to EMPLOYEE-NAME (which in this example would be undesirable).

Such coding practices are not recommended, and this extension is provided only for compatibility.

[End of IBM Extension]

Conceptual Example

Figure 5 on page 129 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.

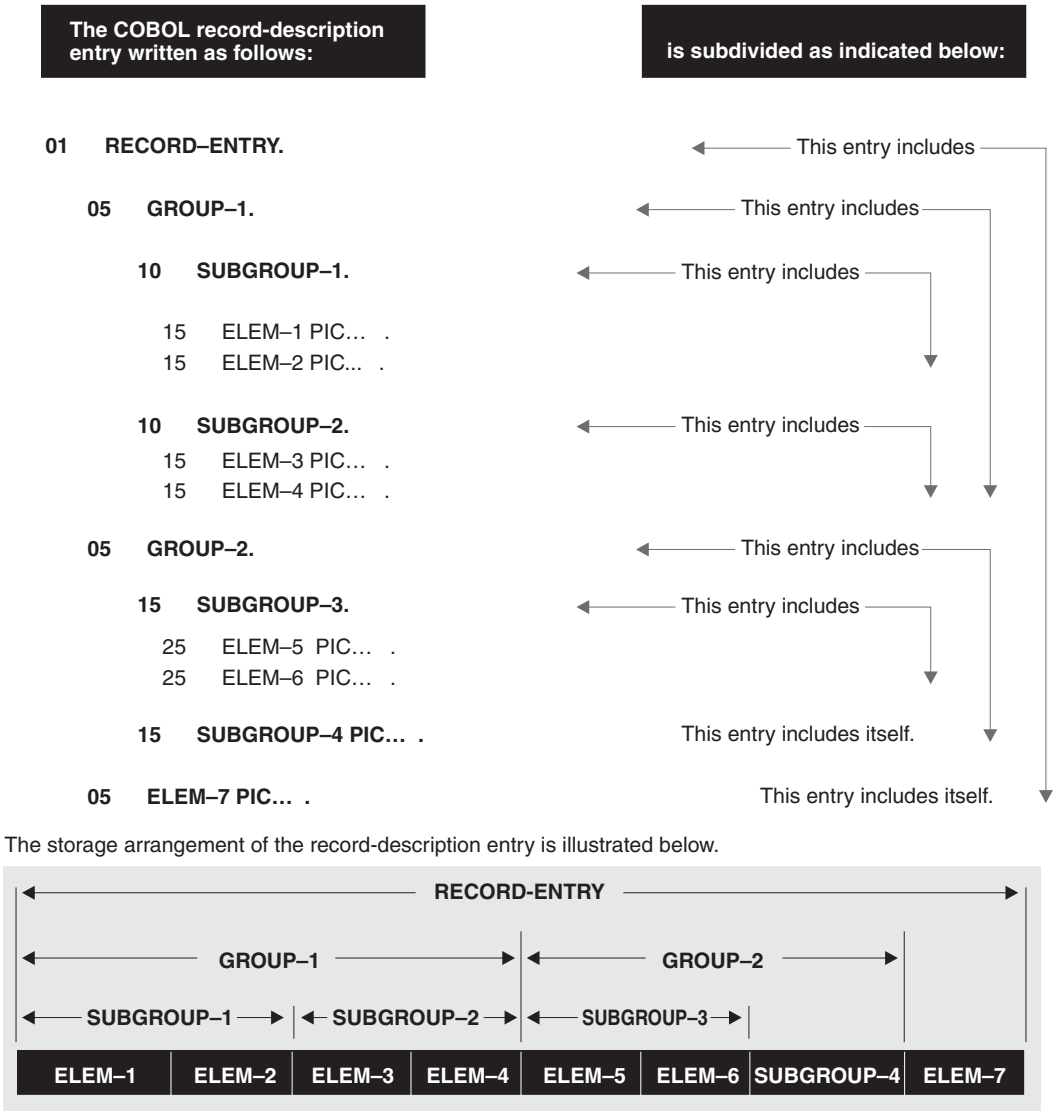


Figure 5. Levels in a Record Description

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 only a RENAMES clause; such items regroup previously defined data items. (For details, see "RENAMES Clause" on page 192.)

77
Identifies data item description entries — independent Working-Storage, Local-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. The condition-name entry must contain only a VALUE clause. (For details, see "VALUE Clause" on page 212.)

Note: Level-77 and level-01 entries in the Working-Storage, Local-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

Most data used in a COBOL program can be divided into classes and categories, except pointers, procedure-pointers, and index data items. 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 9 on page 130 shows the relationship of data classes and categories.

The data category of an item is determined by its PICTURE character-string, BLANK WHEN ZERO, and USAGE attribute. For details, see “Data Categories and PICTURE Rules” on page 181.

[IBM Extension]

The data category of an item can also be determined by its FORMAT clause. A FORMAT clause defines category date, time, and timestamp items.

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]

Every data item that is an intrinsic function is an elementary item, and belongs to the category alphanumeric, numeric, DBCS, national, boolean, date, time, or timestamp and to the corresponding class; the category of each intrinsic function is determined by the definition of the function.

Classes and Categories of Data

Table 9. Class, category, and usage of data items

Class of elementary data items[2]	**Category**	**Usage**
Alphabetic	Alphabetic	DISPLAY
Alphanumeric	Numeric Edited Alphanumeric Edited Alphanumeric	DISPLAY

Table 9. Class, category, and usage of data items (continued)

Class of elementary data items[2]	Category	Usage
Numeric	Numeric	DISPLAY (type zoned decimal) NATIONAL (type national decimal) PACKED-DECIMAL (type internal decimal) COMP-3 (type internal decimal) BINARY COMP COMP-4 COMP-5
	Internal Floating-Point[1]	COMP-1 COMP-2
	External Floating-Point[1]	DISPLAY
Boolean[1]	Boolean[1]	DISPLAY
DBCS[1]	DBCS[1] DBCS-edited[1]	DISPLAY-1
National[1]	National[1]	NATIONAL
Date-Time[1]	Date[1] Time[1]	DISPLAY PACKED-DECIMAL
	Timestamp[1]	DISPLAY

1. IBM Extension
2. The class of group items is alphanumeric for all categories.

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 339).

Numeric

1. The data is aligned on the **assumed decimal point** and if necessary, truncated or padded with zeros. (An assumed decimal point - PICTURE character P or V - 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.

However, if the LOCALE phrase of the PICTURE clause is specified in its data description entry, alignment and zero-fill or truncation takes place as described in "LOCALE Phrase" on page 175.

[IBM Extension] Internal Floating-point

A decimal point is assumed immediately to the left of the field. The data is aligned then on the leftmost digit position following the decimal point, with the exponent adjusted accordingly.

[End of IBM Extension]

[IBM Extension] External Floating-point

The data is aligned on the leftmost digit position; the exponent is adjusted accordingly.

[End of IBM Extension]

Alphanumeric, Alphanumeric-edited, Alphabetic

- The data is aligned at the leftmost character position, and (if necessary) truncated or padded with spaces to the right.
- If the JUSTIFIED clause is specified for this receiving item, the above rule is modified as described in "JUSTIFIED Clause" on page 163.

[IBM Extension]

- For a DBCS receiving item the data is aligned at the leftmost character position,and (if necessary) truncated or padded with DBCS spaces to the right.
- If the JUSTIFIED clause was specified for the DBCS receiving item, the above rule is modified as described in "JUSTIFIED Clause" on page 163.
- For a national receiving item the data is aligned at the leftmost character position,and (if necessary) truncated or padded with national (UCS-2) spaces to the right.
- If the JUSTIFIED clause was specified for the national receiving item, the above rule is modified as described in "JUSTIFIED Clause" on page 163.

[End of IBM Extension]

[IBM Extension] Date, Time, and Timestamp

1. For class date-time items with a USAGE of DISPLAY, data is aligned at the leftmost character position, and (if necessary) padded with spaces to the right.
2. For class date-time items with a USAGE of PACKED-DECIMAL, data is aligned at the rightmost digit position, and (if necessary) padded with zeros to the left.

[End of IBM Extension]

Standard Data Format

For the ILE COBOL 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.

For items described with USAGE DISPLAY (categories alphabetic, alphanumeric, alphanumeric-edited, numeric-edited, numeric, and external floating-point), 1 byte of storage is reserved for each character position described by the item's PICTURE character-string and SIGN IS SEPARATE clause (if applicable).

[IBM Extension]

For items described with USAGE DISPLAY-1 (category DBCS), 2 bytes of storage are reserved for each character position described by the item's PICTURE character-string.

For items described with USAGE NATIONAL (categories national and numeric), 2 bytes of storage are reserved for each character position described by the item's PICTURE character-string and SIGN IS SEPARATE clause (if signed numeric).

The size of an elementary item with a PICTURE clause that includes the LOCALE phrase is determined from integer-1 of the SIZE phrase.

For internal floating-point items, the size of the item in storage is determined by its USAGE clause. USAGE COMPUTATIONAL-1 reserves 4 bytes of storage for the item; USAGE COMPUTATIONAL-2 reserves 8 bytes of storage.

The size of an elementary item of class date-time is determined from the FORMAT literal or from the integer in the SIZE phrase.

[End of IBM Extension]

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 201.

Signed Data

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

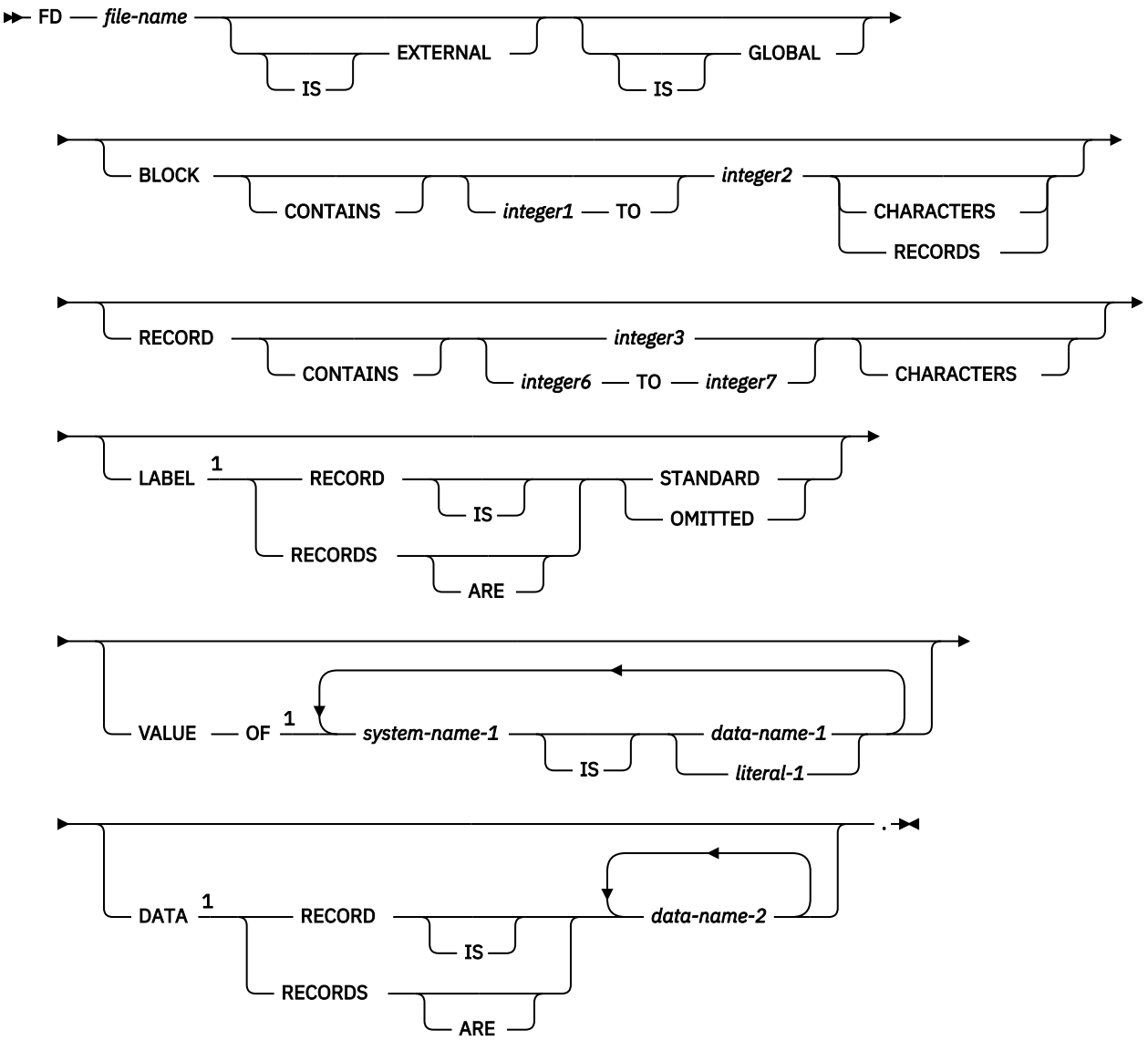
Operational signs (+, -) are associated with signed numeric items, and indicate their algebraic properties. The internal representation of an algebraic sign depends on the item's USAGE clause, and, optionally, upon its SIGN clause. Zero is considered a unique value, regardless of the operational sign.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

Editing signs 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.

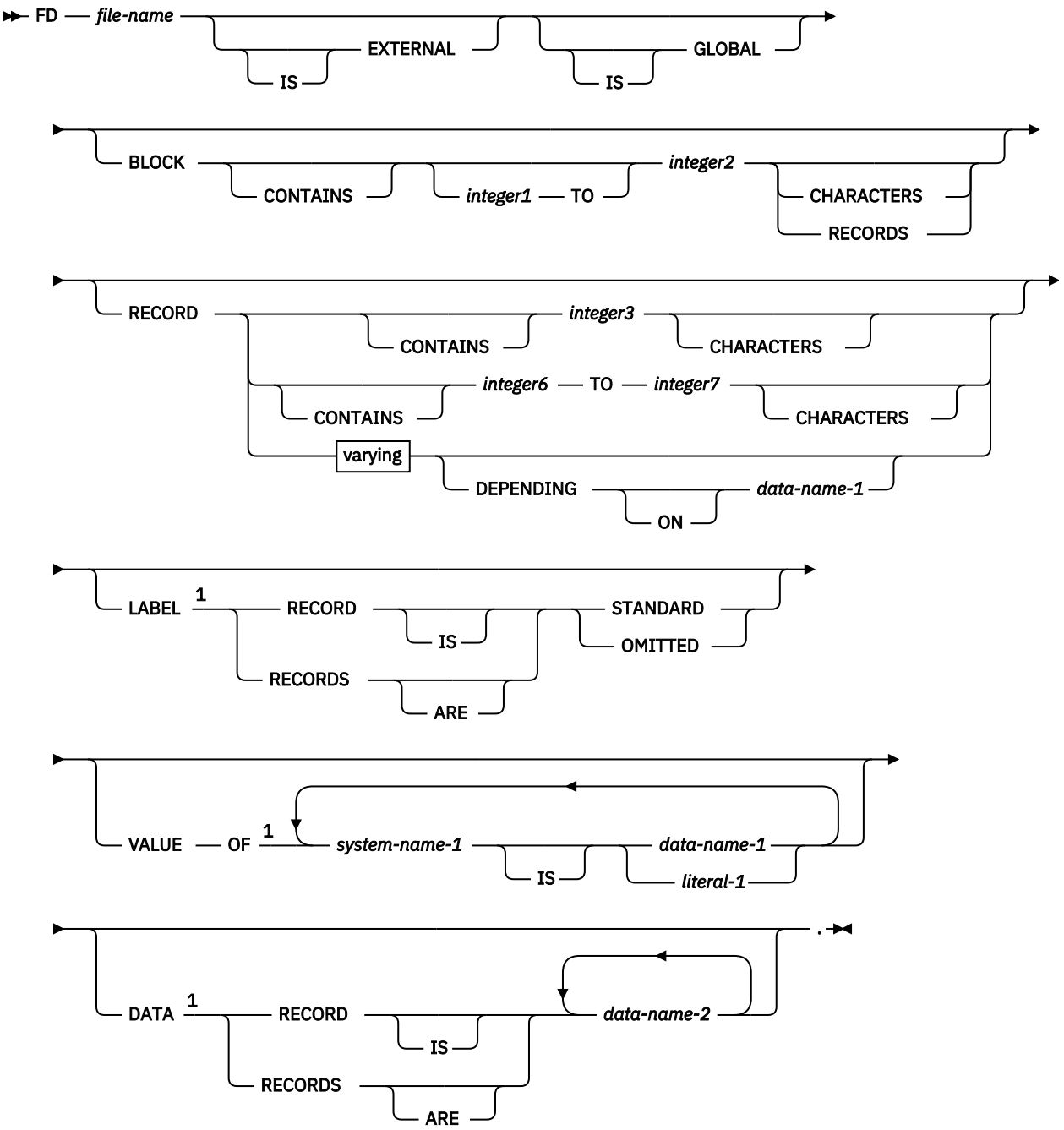
File Description Entry - Format 1 - Sequential File



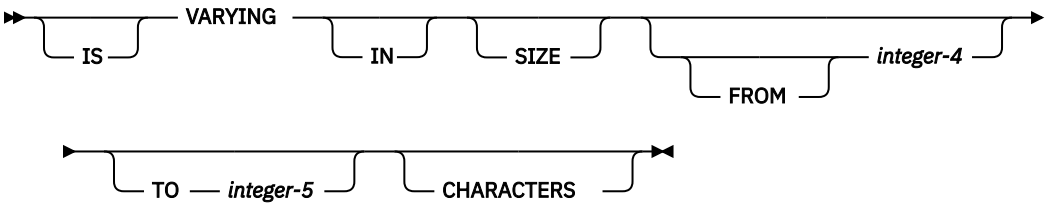
File Description Entry - Format 1a - Formatfile, Database

Notes:

[1] Syntax-checked only.



varying

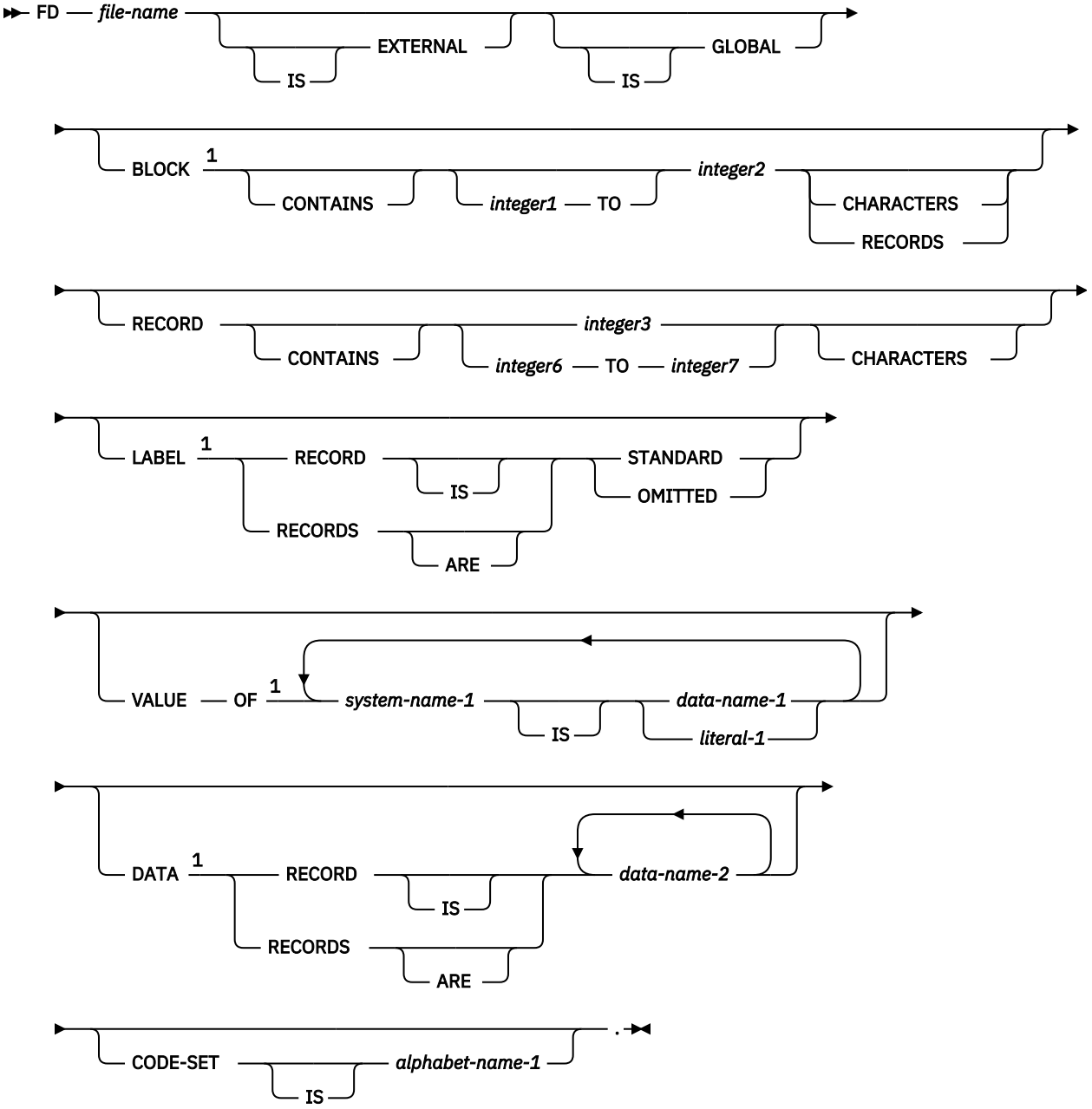


File Description Entry - Format 1b - Disk

Notes:

[1] Syntax-checked only.

File Description Entry - Format 2 - Diskette File

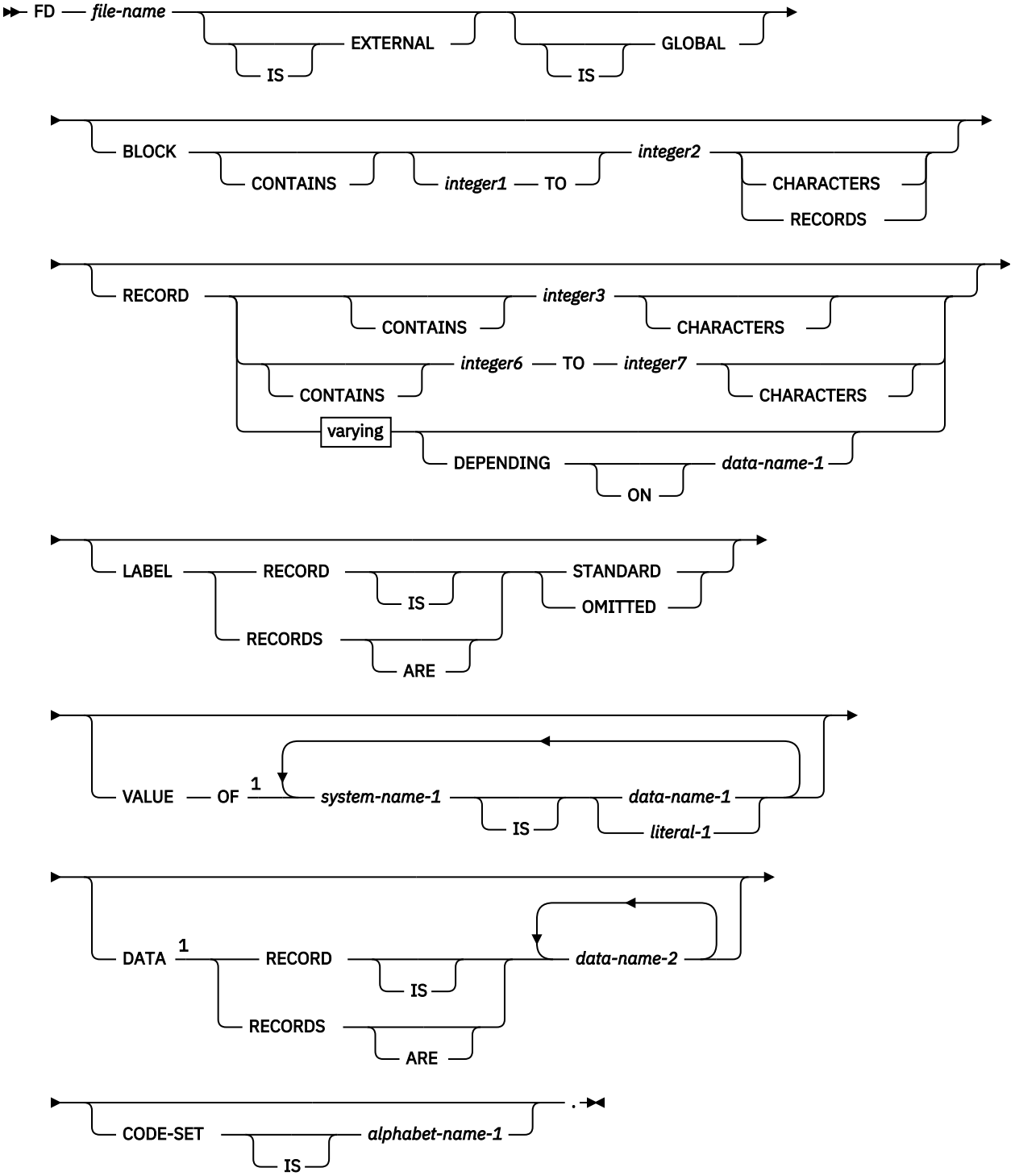


File Description Entry - Format 2 - Diskette

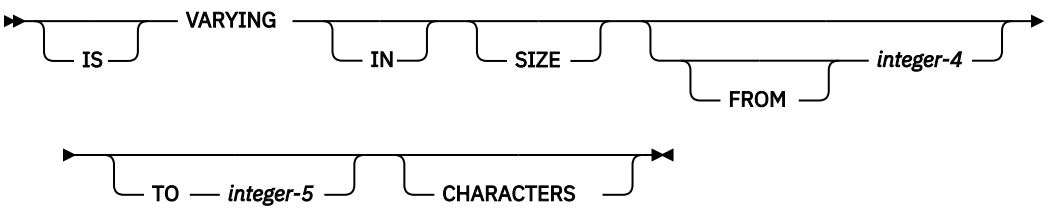
Notes:

[1] Syntax-checked only.

File Description Entry - Format 3 - Tapefile



varying

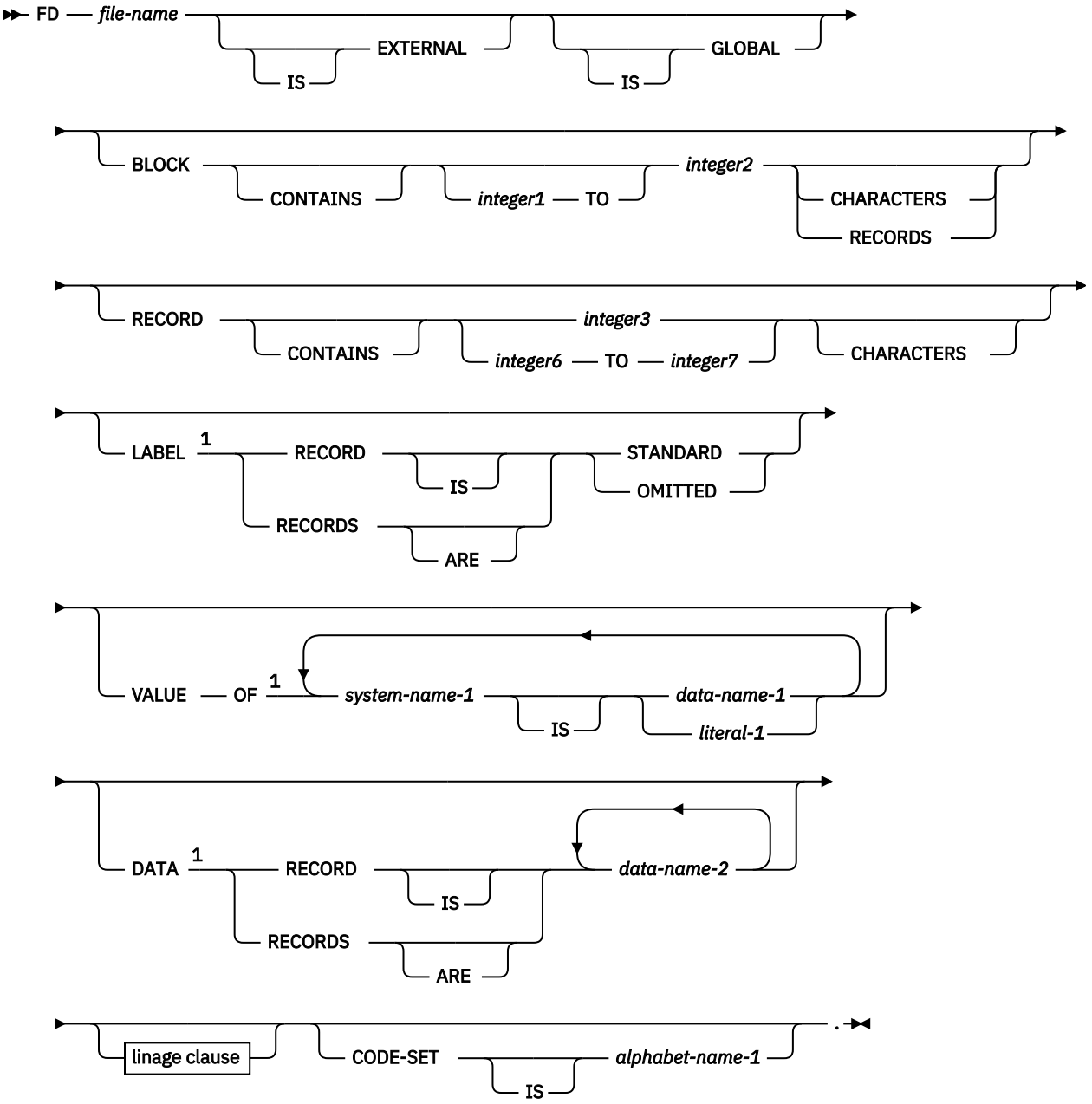


File Description Entry - Format 3 - Tapefile

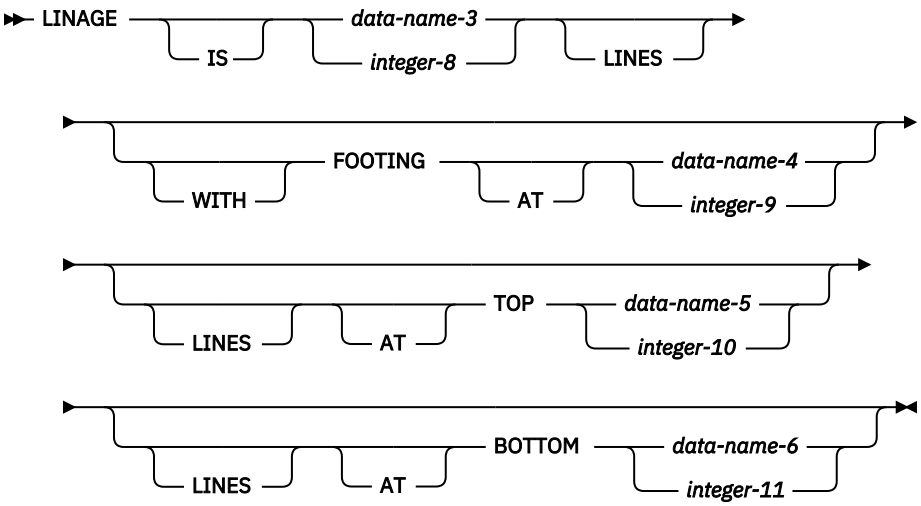
Notes:

[1] Syntax-checked only.

File Description Entry - Format 4 - Printer File



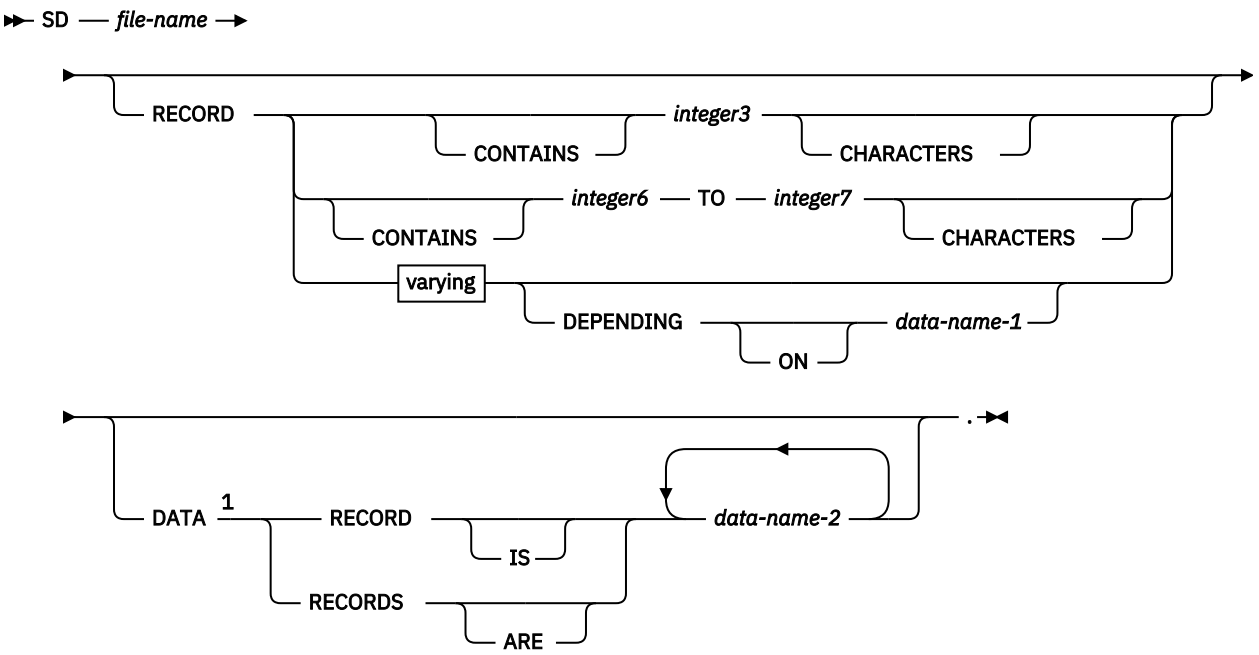
linage clause



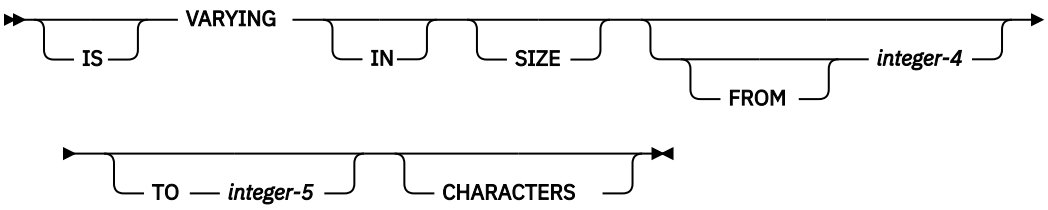
File Description Entry - Format 4 - Printer
Notes:

[1] Syntax-checked only.

Sort Description Entry - Format 5 - Sort or Merge Files



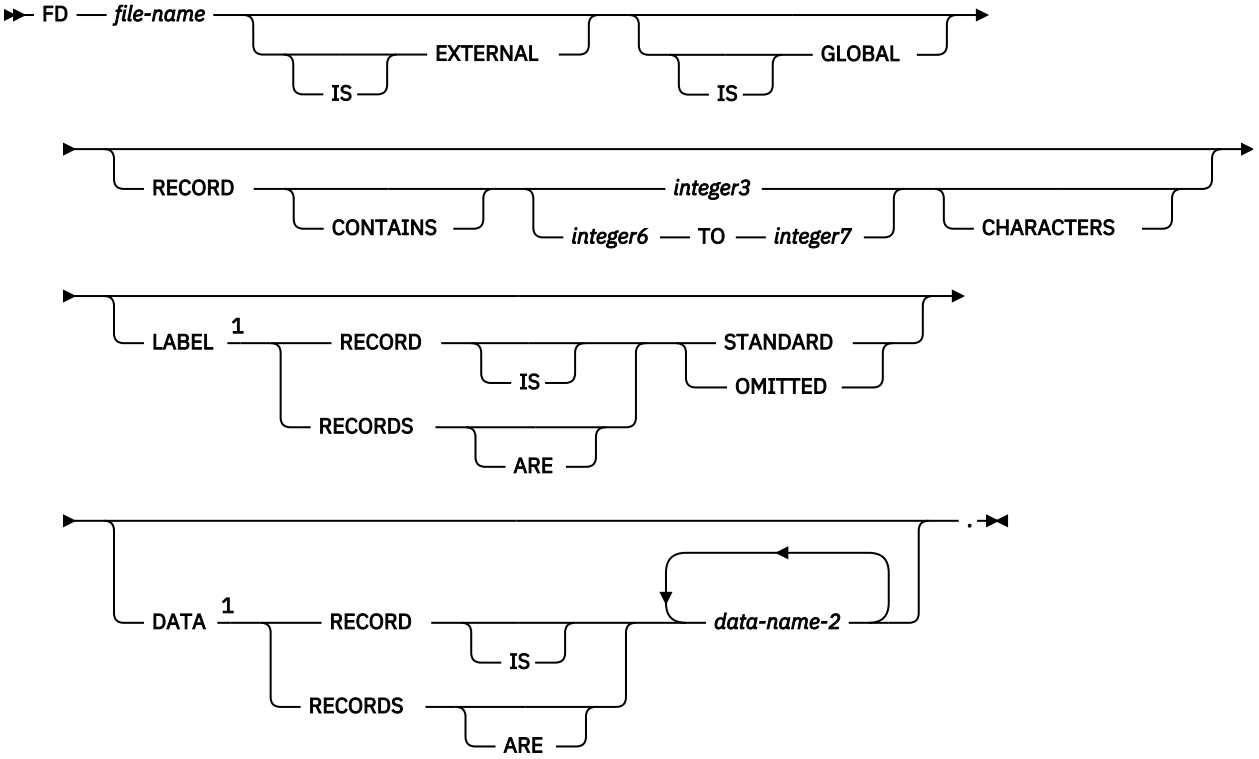
varying



File Description Entry - Format 5 - Sort/Merge
Notes:

[1] Syntax-checked only.

[IBM Extension] File Description Entry - Format 6 - Transaction Files



File Description Entry - Format 6 - Transaction

Notes:

[1] Syntax-checked only.

[End of IBM Extension]

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. The last clause of an FD entry must be immediately followed by a separator period.
- For each sort or merge file, the File Section must contain an SD entry. The last clause of an SD entry must be immediately followed by a separator period.

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.

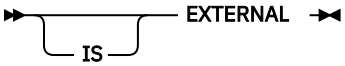
One or more record description entries must follow the file-name. A record description entry may describe a type-name. Each entry, which is not a type-name, implies a redefinition of the same storage area.

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

EXTERNAL Clause

The EXTERNAL clause specifies that a file connector is external.

EXTERNAL Clause - Format



In a run unit, there is only one representation of an external file; an external file can be referenced by any COBOL program in the run unit that describes the file.

In the File Section, the EXTERNAL clause can be specified only in file description entries. The records appearing in the file description entry need not have the same name in corresponding external file description entries. In addition, the number of such records need not be the same in corresponding file description entries. However, the maximum record length for corresponding external file description entries must be the same.

For EXTERNAL files, the value of all BLOCK CONTAINS clauses of corresponding EXTERNAL files must match within the run unit. This conformance is in terms of character positions only - not upon whether the value was specified as CHARACTERS or as RECORDS.

All file description entries in the run unit that are associated with an external file connector must have:

- A LINAGE clause, if any file description entry has a LINAGE clause.
- The same corresponding values for integer-8, integer-9, integer-10, and integer-11, if specified.
- The same corresponding external data items referenced by data-name-3, data-name-4, data-name-5, and data-name-6.

Use of the EXTERNAL clause does not imply that the associated file-name is a global name.

The TYPEDEF clause cannot be specified in the same data description entry as the EXTERNAL clause, however, the TYPE clause can.

Considerations for External Files

In general, all definitions of an external file should be identical. If there is a mismatch, the program will fail at start up when the definitions are compared. The following attributes of external files are compared:

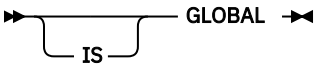
- If any of the definitions corresponding to the file are externally described (for example, by using Format 2 of the COPY statement), all other definitions must also be externally described. The level check information associated with all definitions must match.
- The name specified on the ASSIGN TO clause must be the same for all definitions. This includes the device type.
- The ORGANIZATION and ACCESS modes must be the same for all definitions of the file.
- The OPTIONAL phrase, if specified, must be specified for all definitions of the file.
- The external data item specified for the RELATIVE KEY phrase must be in the same physical location and occupy the same number of bytes for all definitions of the file.
- The location of the record key within the associated record must be the same for all definitions of the file.
- The blocking information associated with the file must be the same for all file definitions. This includes whether blocking is to be performed and the size of the block.
- The values for the maximum or minimum number of characters on the RECORD clause must be the same for all definitions of the file.
- The character set specified on the CODE-SET clause must be the same for all definitions of the file.
- The value specified for the DUPLICATES phrase must be the same for all definitions of the file.
- All of the values specified for the LINAGE clause must be the same for each definition of the file.
- The specification of the attribute for the ASSIGN clause (separate indicators) must be the same for all definitions of the file.
- The specification for the COMMITMENT CONTROL clause must be the same for all definitions of the file.

- The specification for the *DUPKEYCHK or the *INZDLT compile-time option must be the same for all modules containing definitions of the file.
- The specifications of the CCSID parameter for the CRTCBLMOD or CRTBNDCBL command must be the same for all modules containing definitions of the file.

GLOBAL Clause

The GLOBAL clause specifies that the file connector named by a file-name is a global name.

GLOBAL Clause - Format



A global file-name is available to the program that declares it and to every program that is contained directly or indirectly in that program.

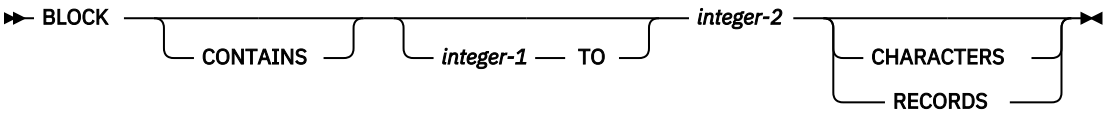
A file-name is global if the GLOBAL clause is specified in the file description entry for that file-name. A record-name is global if the GLOBAL clause is specified in the record description entry by which the record-name is declared or, in the case of record description entries in the File Section, if the GLOBAL clause is specified in the file description entry for the file-name associated with the record description entry. Such record description entries may describe a type-name.

The GLOBAL clause can be specified in the same data description entry as the TYPEDEF clause. The scope of the clause applies to the type-name only, and not to any data items which are defined using a global type-name with a TYPE clause.

BLOCK CONTAINS Clause

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

BLOCK CONTAINS Clause - Format



This clause can be omitted when each physical record contains only one complete logical record.

This clause is syntax checked for FORMATFILE and printer files. To activate this clause for other types of files, use OPTION parameter value *BLK, or PROCESS option BLK.

integer-1, integer-2
Must be nonzero unsigned integers. They specify the number of characters or records. If integer-2 is zero, the system will determine the blocking size.

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.

In general, the length of a variable length record on a RELEASE, REWRITE, or WRITE statement is determined by data-name-1, if specified. If data-name-1 is not specified and the record description does not contain a table, the length is the number of characters in the record description. If data-name-1 is not specified and the record contains a table, the length is the sum of the fixed part of the record and the current length of the table.

When variable length records are used for disk files, the BLOCK CONTAINS clause specifies the size of the block. The size of the actual record is contained in data-name-1 after a READ operation. To WRITE a variable length record, data-name-1 must be set to the length of the record.

For tape files, 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 32 764.

When variable records are used for tape files, 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 32 767; maximum block size is 32 767. 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.

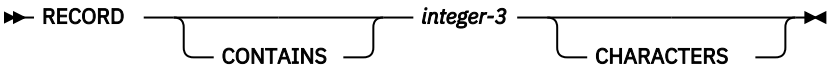
RECORD Clause

The RECORD clause specifies the number of character positions for fixed-length or variable-length records.

RECORD clause - Format 1

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

RECORD Clause - Format 1



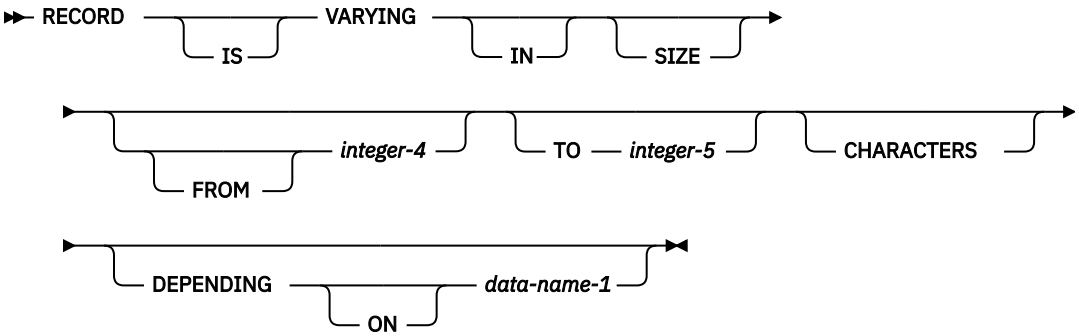
integer-3
Must be an unsigned integer that specifies the number of character positions contained in each record in the file.

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

RECORD clause - Format 2

Format 2 is the recommended format when dealing with variable records.

RECORD Clause - Format 2



integer-4
Specifies the minimum number of character positions to be contained in any record of the file. If integer-4 is not specified, the minimum number of character positions to be contained in any record of the file is equal to the smallest number of character positions described for a record in that file. If specified, integer-4 must be nonzero and less than integer-5.

integer-5
Specifies the maximum number of character positions in any record of the file. If integer-5 is not specified, the maximum number of character positions in any record of the file is equal to the greatest number of character positions described for a record in that file.

data-name-1
Must be an elementary unsigned integer.

If data-name-1 is specified:

- The number of character positions in the record must be placed into the data item referenced by data-name-1 before any RELEASE, REWRITE, or WRITE statement is executed for the file.
- The execution of a DELETE, RELEASE, REWRITE, START, or WRITE statement or the unsuccessful execution of a READ or RETURN statement does not alter the contents of the data item referenced by data-name-1.
- After the successful execution of a READ or RETURN statement for the file, the contents of the data item referenced by data-name-1 indicate the number of character positions in the record just read.

The number of character positions associated with a record description is determined by the sum of the number of character positions in all elementary data items (excluding redefinitions and renamings), plus any implicit FILLER due to synchronization.

If a table is specified, the minimum and maximum number of table elements described in the record are used in the summation above, to determine the minimum and maximum number of character positions associated with the record description.

If the number of character positions in the logical record to be written is less than integer-4 or greater than integer-5, the output statement is unsuccessful and, except during execution of a RELEASE statement, the associated I-O status key is set to a value indicating the cause of the condition.

During the execution of a RELEASE, REWRITE, or WRITE statement, the number of character positions in the record is determined by the following conditions:

- If data-name-1 is specified, by the content of the data item referenced by data-name-1.
- If data-name-1 is not specified and the record does not contain a variable occurrence data item, by the number of character positions in the record.
- If data-name-1 is not specified and the record contains a variable occurrence data item, by the sum of the fixed portion and that portion of the table described by the number of occurrences at the time of execution of the output statement.

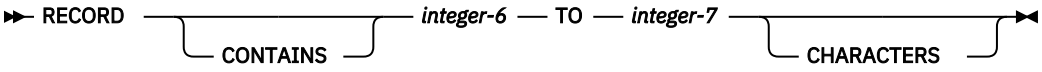
During the execution of a READ ... INTO or RETURN ... INTO statement, the number of character positions in the current record that participate as the sending data items in the implicit MOVE statement is determined by the following conditions:

- If data-name-1 is specified, by the content of the data item referenced by data-name-1.
- If data-name-1 is not specified, by the value that would have been moved into the data item referenced by data-name-1 had data-name-1 been specified.

RECORD clause - Format 3

Format 3 specifies the number of character positions for either fixed or variable length records. (The latter is only applicable to tape files.)

RECORD Clause - Format 3



For Tape Files

In this case, the records are variable, and the following descriptions apply.

integer-6, integer-7
Must be unsigned integers. Integer-6 specifies the size of the smallest data record, and integer-7 specifies the size of the largest data record.

For All Other Files

If Format 3 is used for non-tape files, the records are treated as fixed length records the size of the largest data record. 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 32 767.

General Considerations for all Formats

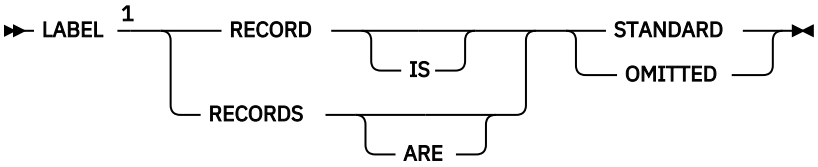
When the RECORD clause is used, the record size must be specified as the number of character positions needed to store the record internally. That is, it must specify the number of bytes occupied internally by the characters of the record, not the number of characters used to represent the data item within the record. The size of a record is determined according to the rules for obtaining the size of a group item.

When the RECORD clause is omitted, the compiler determines the record lengths from the record descriptions. When one of the entries within a record description contains an OCCURS DEPENDING ON clause, the compiler uses the maximum value of the variable-length item to calculate the number of character positions needed to store the record internally.

If the associated file connector is an external file connector, all file description entries in the run unit that are associated with that file connector must specify the same maximum number of character positions.

LABEL RECORDS Clause

The LABEL RECORDS clause indicates the presence or absence of labels. This clause is only significant for FD - Format 3 (TAPEFILE). For all other FD formats, this clause is syntax checked only, then treated as documentation.



LABEL RECORDS Clause - Format

Notes:

1 Syntax-checked only.

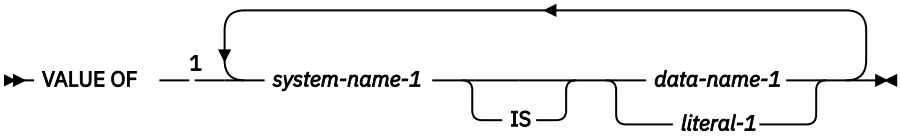
If it is not specified for a file, label records for that file must conform to the system label specifications.

STANDARD
Labels conforming to system specifications exist for this file.

OMITTED
No labels exist for this file.

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.



VALUE OF Clause - Format

Notes:

1 Syntax-checked only.

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

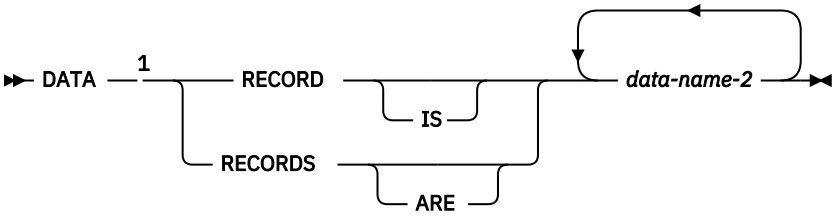
literal-1
Can be any literal.

[IBM Extension] Floating-point literals cannot be used. [End of IBM Extension]

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 and it serves only as documentation for the names of data records associated with this file.



DATA RECORDS Clause - Format

Notes:

1 Syntax-checked only.

data-name-2
The names of record description entries associated with this file. Data-name-2 must not be a type-name.

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.

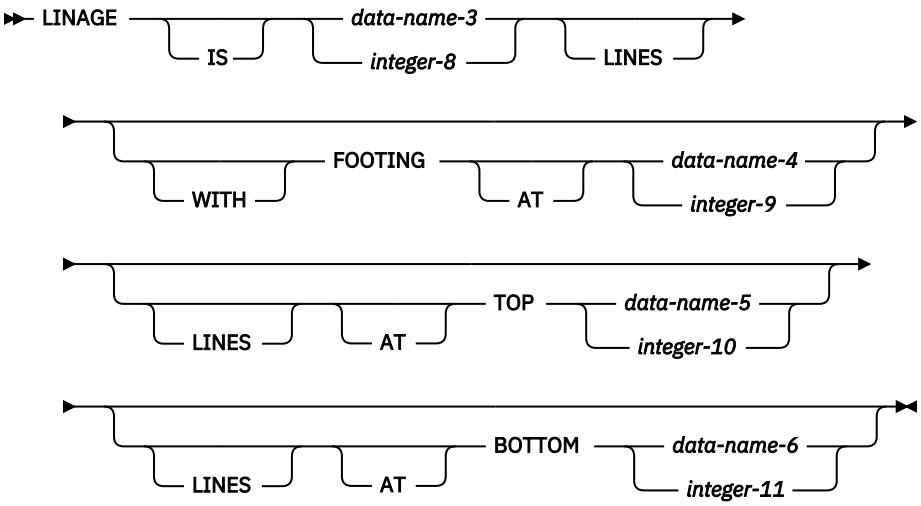
LINAGE Clause

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 don't necessarily have to be of the same size.)

The LINAGE clause can be specified only for files assigned to the device PRINTER. See "ASSIGN Clause" on page 102.

IBM i printer files offer a number of powerful features through DDS that can be used to advantage. Such files are declared in ILE COBOL programs as FORMATFILE. For more information on printer files, see the *Db2 for i* section of the *Database and File Systems* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

LINAGE Clause - Format



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.

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

data-name-3, integer-8
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-9 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-10 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-11 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 148 illustrates the use of each phrase of the LINAGE clause.

Illustration of LINAGE clause phrases

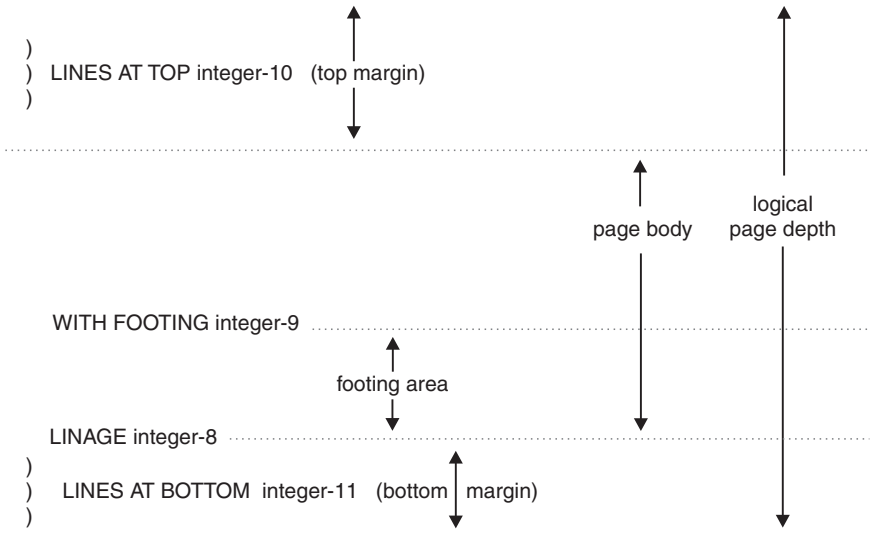


Figure 6. LINAGE Clause Phrases

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-9 is equal to the LINAGE value of data-name-3 or integer-8, one line (the last line of the logical page) is available for footing information.

If the FOOTING phrase is omitted, its assumed value is equal to that of the page body (integer-8 or data-name-3).

At the time an OPEN OUTPUT statement is executed, the values of integer-8, integer-9, integer-10, and integer-11, 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 on page 148. 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

A separate LINAGE-COUNTER special register is generated for each FD entry containing a LINAGE clause (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-8, LINAGE-COUNTER is a binary item large enough to hold the binary representation of integer-8.

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 431.)

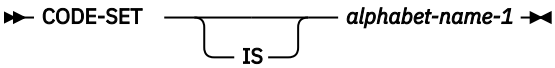
If the file description for a sequential file contains the LINAGE clause and the EXTERNAL clause, the LINAGE-COUNTER data item is an external data item. Similarly, if it contains the LINAGE and GLOBAL clauses, the LINAGE-COUNTER data item is a global data item.

You can specify the LINAGE-COUNTER special register wherever an integer argument to a function is allowed.

CODE-SET Clause

The CODE-SET clause specifies the character code used to represent data on DISKETTE and TAPEFILE.

CODE-SET Clause - Format



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

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.

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), NATIVE (for EBCDIC-encoded files), or EBCDIC (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.

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 IBM i 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. For more information on these commands, see the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

[End of IBM Extension]

Data Division—Data Description Entry

A data description entry specifies the characteristics of a data item. Data items have **attributes**, which may be either **implicit** (default values), or **explicit**.

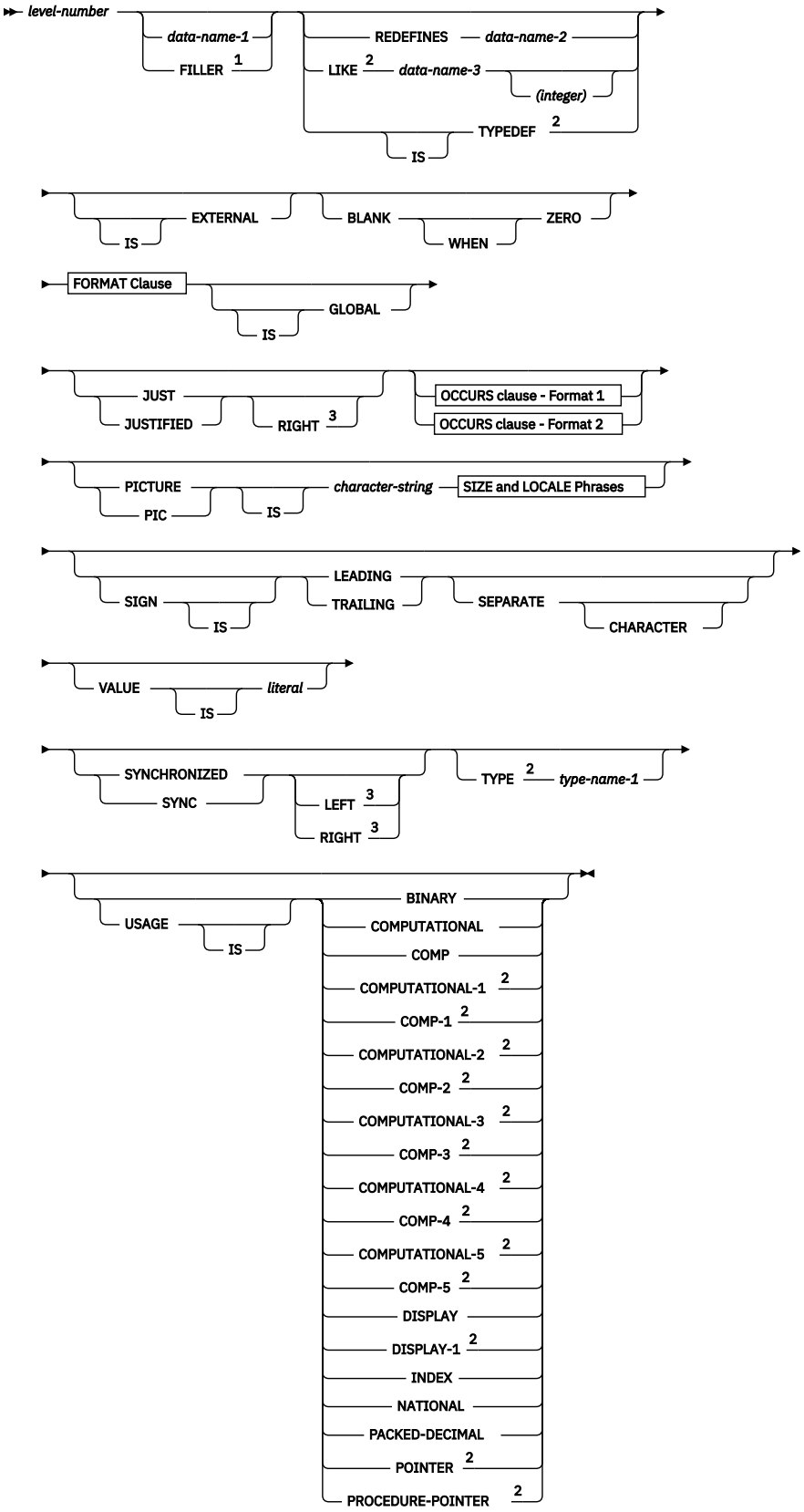
This section describes the coding of data description entries and record description entries (which are sets of data description entries). The single term **data description entry** is used in this section to refer to data and record description entries.

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 four general formats.

Format 1

Format 1 is used for data description entries in all Data Division sections. Level-number in this format can be any number from 01-49, as well as 77.



Data Description Entry - General Format 1

Notes:

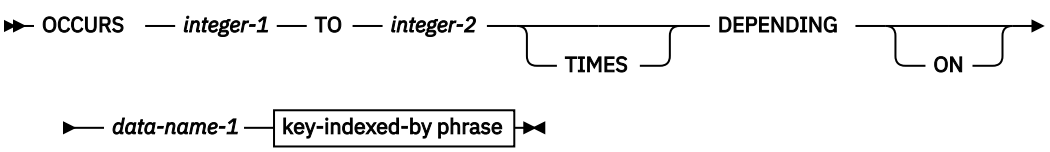
[1] Cannot be used with the TYPEDEF clause.

[2] IBM Extension

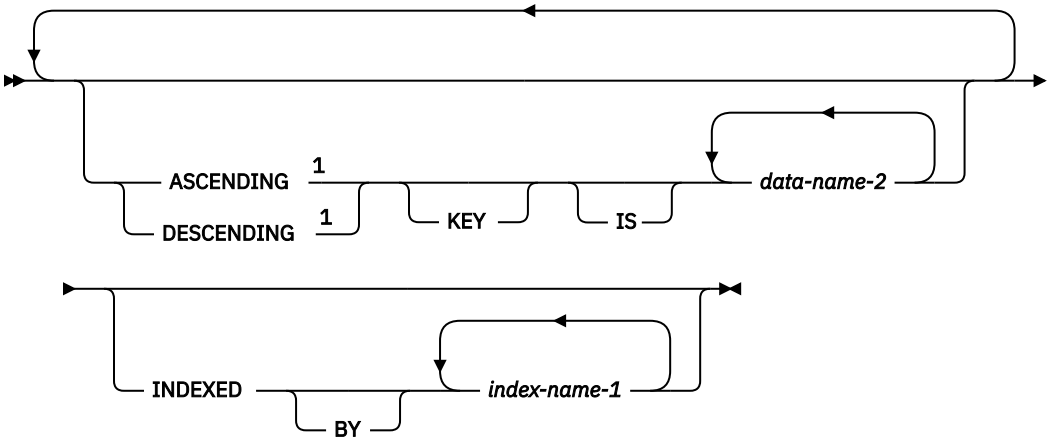
[3] Syntax-checked only

OCCURS clause - Format 1

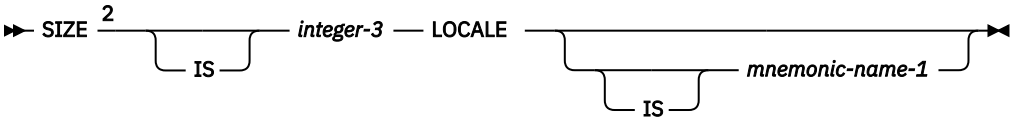
OCCURS clause - Format 2



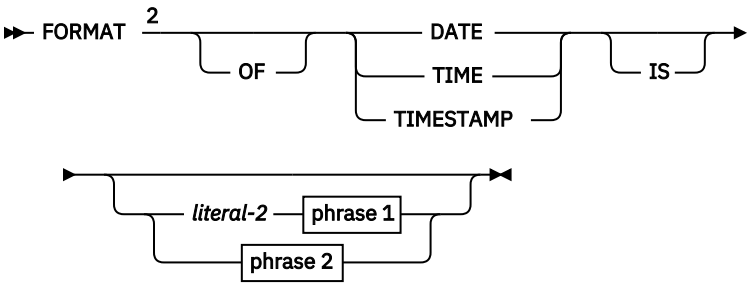
key-indexed-by phrase



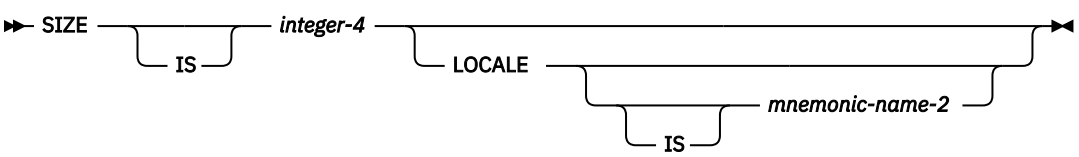
Size and Locale Phrases



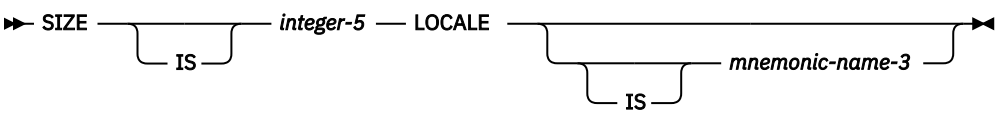
FORMAT Clause



phrase 1



phrase 2



Data Description Entry - General Format 1 (continued)
Notes:

[1] Cannot be used with boolean data type
[2] IBM Extension

The clauses may be written in any order with three exceptions:

- If data-name or FILLER is specified, it must immediately follow the level-number.
- 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 data item being described is treated as though FILLER has been specified.
- When specified, the TYPEDEF clause must be the first entry following data-name-1. The TYPEDEF clause cannot be specified with FILLER. The TYPEDEF clause and the REDEFINES clause cannot both be specified for data-name-1.

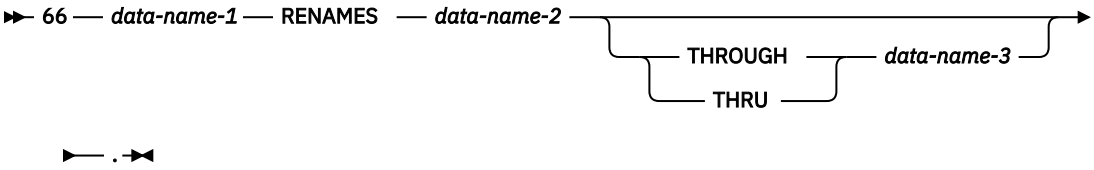
Not all clauses are compatible with each other. For details, see the descriptions of the individual clauses.

Clauses must be separated by a space, a separator comma, or a separator semicolon.

Format 2

Format 2 regroups previously defined items.

Data Description Entry - 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.

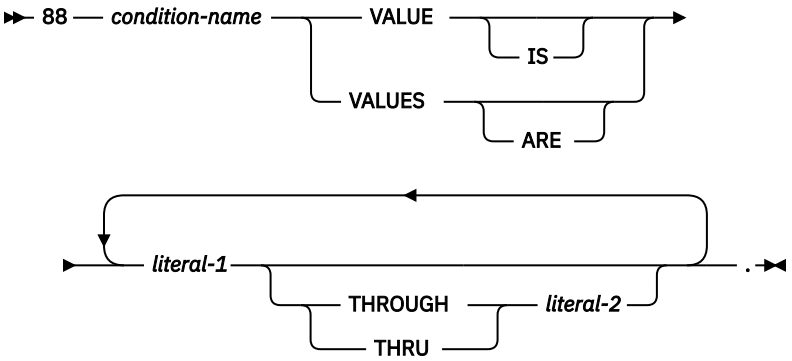
All level-66 entries associated with one record must immediately follow the last data description entry in that record.

Details are contained in “RENAMES Clause” on page 192.

Format 3

Format 3 describes condition-names.

Data Description Entry - 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.

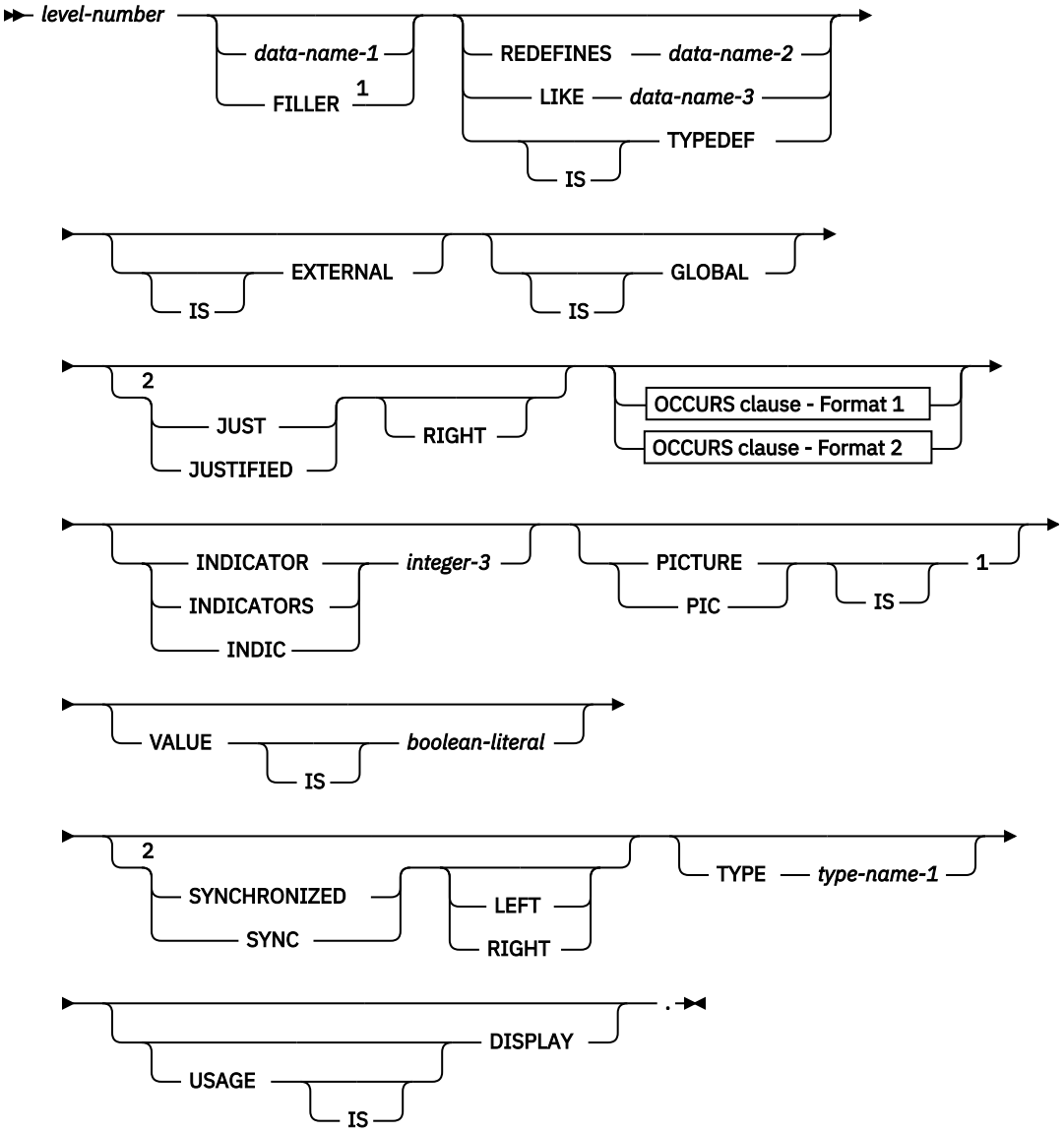
A **conditional variable** is a data item that can assume one or more values, that can, in turn, be associated with a condition-name.

Format 3 can be used to describe both elementary and group items. Further information on condition-name entries can be found under "Condition-Name Condition" on page 227.

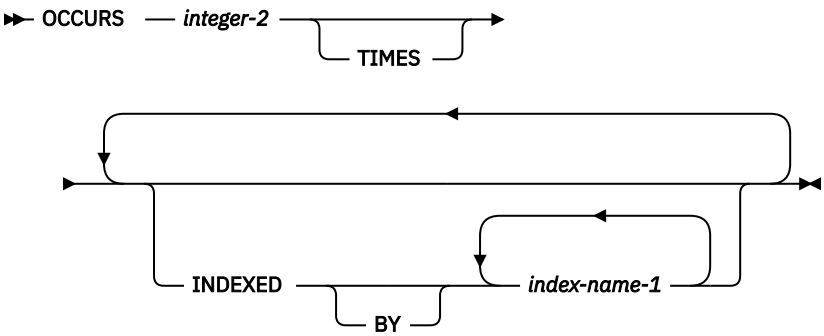
[IBM Extension] Format 4

This format describes Boolean data. **Boolean data items** are items that are limited to a value of 1 or 0.

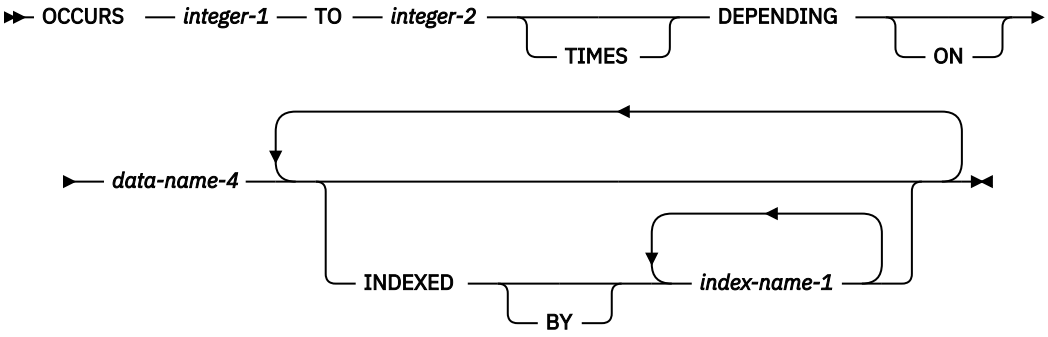
Note: When you use indicators in a COBOL program, you must describe them as Boolean data items using the data description entry for Boolean data.



OCCURS clause - Format 1



OCCURS clause - Format 2



Data Description Entry - Format 4 - Boolean Data

Notes:

[1] Cannot be used with the TYPEDEF clause.

[2] Syntax-checked only

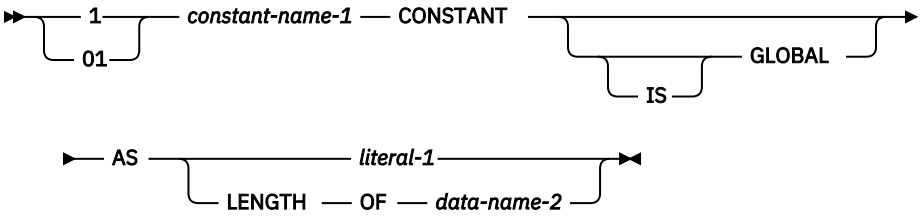
The special considerations for the clauses used with the Boolean data follow. All other rules for clauses are the same as those for other data.

[End of IBM Extension]

Format 5

[IBM Extension] Format 5 describes constant-names. Constant-name can only be described as a Level-01 entry. Further information on constant-name can be found under “CONSTANT Clause” on page 155. [End of IBM Extension]

[IBM Extension]



[End of IBM Extension]

[IBM Extension] CONSTANT Clause

The CONSTANT clause is used to associate a constant name with a literal. The constant name can then be used in place of a literal. The CONSTANT clause can only be specified for level 01 entries for elementary constant name. The CONSTANT clause can also be defined as another previously defined constant name.

A Constant name needs to be defined in a CONSTANT clause before its use. It can be used in Data Division and Procedure Division where literal or integer is allowed, except in the compiler-directing statements, such as COPY statement and TITLE statement.

Constant-name-1 may be used anywhere that a format specifies a literal of the class and category of constant-name-1. The class and category of constant-name-1 is the same as that of literal-1 or is an integer if LENGTH OF phrase is specified. If constant-name-1 is an integer, it may also be used to specify repetition in a picture string.

Literal-1 cannot be a figurative constant.

If the LENGTH OF phrase is specified, the value of constant-name-1 is determined as specified in the LENGTH intrinsic function with the exception that when data-name-2 is a variable-length data item described with the OCCURS DEPENDING ON clause, the maximum size of the data item is used.

[End of IBM Extension]

LIKE Clause

The length of the data item cannot be changed using this clause.

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.

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.

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 type-name, a renaming or redefining 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.

level-number

Level-numbers must be followed either by a separator period; or by a space followed by its associated data-name-1, FILLER, or appropriate data description clause. Level number 01 and 77 must begin in

Area A. Level-number 77 must be followed by a space followed by its associated data-name-1. Level numbers 02 through 49, 66, and 88 may begin in Area A or B.

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 127 and "Standard Data Format" on page 132.

[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-1
Explicitly identifies the data being described.

If specified, data-name-1 identifies a data item used in the program. The data item must be the first word following the level-number.

The data item can be changed during program execution.

Data-name-1 must be specified for:

- Level-66, level-77, and level-88 items
- Entries containing a GLOBAL, EXTERNAL, or TYPEDEF clause
- Record description entries associated with file description entries having GLOBAL or EXTERNAL clauses.

FILLER
Is a data item that is not explicitly referred to in a program. The keyword FILLER is optional. If specified, FILLER must be the first word following the level-number.

The keyword 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 or a type-name.

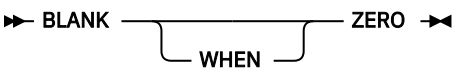
In a MOVE CORRESPONDING statement, or in an ADD CORRESPONDING or SUBTRACT CORRESPONDING statement, FILLER items are ignored. In an INITIALIZE statement, elementary FILLER items are ignored.

If data-name-1 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.

BLANK WHEN ZERO Clause - 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 an entry containing the PICTURE symbols S or *.

The BLANK WHEN ZERO clause is not allowed with:

- USAGE IS INDEX clause

[IBM Extension]

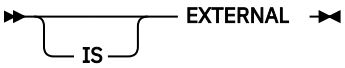
- Items of class date-time
- External or internal floating-point items
- USAGE IS POINTER clause
- items described with the USAGE IS PROCEDURE-POINTER clause
- DBCS items
- National items
- TYPE clause.

[End of IBM Extension]

EXTERNAL Clause

The EXTERNAL clause specifies that the storage associated with a data item is associated with the run unit rather than with any particular program within the run unit.

EXTERNAL Clause - Format



An external data item can be referenced by any program in the run unit that describes the data item. References to an external data item from different programs using separate descriptions of the data item are always to the same data item. In a run unit, there is only one representation of an external data item.

The EXTERNAL clause can be specified in either 01 level entries in the Working-Storage Section or in file description entries. If there are two data description entries with the same data name in the same Data Division, only one entry can contain the EXTERNAL clause. Index-names, condition-names, and renaming (level-66) items in an external data record do not possess the EXTERNAL attribute.

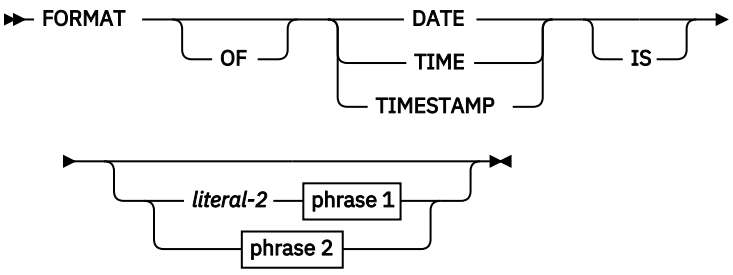
The data contained in the record named by the data-name clause is external and can be accessed and processed by any program in the run unit that describes and, optionally, redefines it. This data is subject to the following rules:

- If two or more programs within a run unit describe the same external data record, each record-name of the associated record description entries must be the same and the records must define the same number of standard data format characters. However, a program that describes an external record can contain a data description entry including the REDEFINES clause that redefines the complete external record, and this complete redefinition need not occur identically in other programs in the run unit.
- Use of the EXTERNAL clause does not imply that the associated data-name is a global name.
- The VALUE clause must not be used in any data description entry which includes, or is subordinate to an entry which includes, the EXTERNAL clause. The VALUE clause can be specified for condition-name entries associated with such data description entries.
- The TYPEDEF clause cannot be specified in the same data description entry as the EXTERNAL clause.

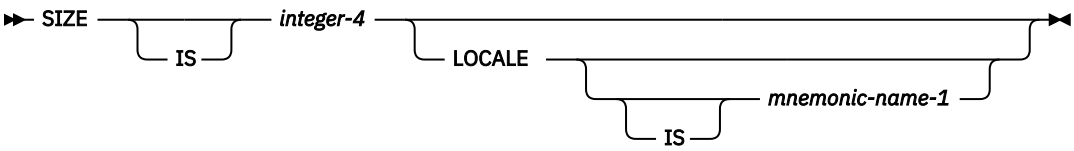
See “Data Reference and Name Scoping” on page 45 for more information.

[IBM Extension] FORMAT Clause

The FORMAT clause specifies the general characteristics and editing requirements of an elementary date, time, or timestamp item.

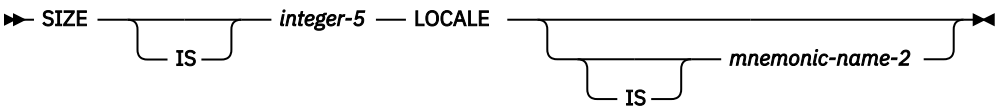


phrase 1



FORMAT Clause - Format

phrase 2



The FORMAT clause must be specified for every elementary date, time, or timestamp item, except the subject of a RENAMES clause.

If the SIZE phrase is not specified for a timestamp item, the size defaults to 26. If it is specified, it must have a value of 19, or a value between 21 and 32.

literal-2 and the LOCALE phrase cannot be specified for a timestamp item. A timestamp has a fixed format, which is dependent on the size of the timestamp item.

- When the SIZE phrase is not specified, the format is equivalent to a literal-2 value of "@Y-%m-%d-%H.%M.%S.@Sm".
- When the SIZE phrase is specified with a value of 19, the format is equivalent to a literal-2 value of "@Y-%m-%d-%H.%M.%S".
- When the SIZE phrase is specified as a value between 21 and 32, the format is equivalent to a literal-2 value of "@Y-%m-%d-%H.%M.%S." followed by the fractional seconds in the timestamp. For example, a timestamp with size 25 could have the value "2014-01-23-01.02.03.12345".

If literal-2 or the LOCALE phrase is not specified for a date or time item, the format of the item is determined from the SPECIAL-NAMES FORMAT clause.

A data item of class date-time cannot be reference modified.

When the FORMAT clause is specified, the following clauses cannot be specified:

- PICTURE clause.
- SIGN clause.
- BLANK WHEN ZERO clause.
- JUSTIFIED clause.
- LIKE clause. A LIKE clause can, however, be used to define the FORMAT of a data item. You cannot change the size of a date, time, or timestamp item with a LIKE clause. When a LIKE clause is referring to a date, time, or timestamp item, a comment is generated with the appropriate FORMAT clause information that is inherited
- TYPE clause.

The following general rules apply:

- A condition-name can be associated with a date-time item. The VALUE clause of the condition-name can be specified with a THRU phrase.
- A SYNCHRONIZED clause is treated as documentation.
- The OCCURS, REDEFINES, and RENAMES clauses can be associated with date, time, or timestamp items.
- If a LIKE clause is specified, a FORMAT clause cannot be specified.
- Any associated VALUE clause must specify a non-numeric literal. The literal is treated exactly as specified; no formatting is done.

literal-2
Specifies the format of a date or time item. Literal-2 must be a non-numeric literal, at least 2 characters long. The contents of literal-2 is made up of separators and conversion specifiers. For a list of valid conversion specifiers, see Table 5 on page 90. For further rules on the contents of literal-2, see the description of the FORMAT clause used in the SPECIAL-NAMES paragraph in "FORMAT Clause" on page 89.

[End of IBM Extension]

SIZE Phrase

For a more detailed description of the SIZE phrase, refer to "SIZE Phrase" on page 91. This section describes the parameters that you specify for this SIZE phrase.

integer-3, integer-4
Integer-3 and integer-4 determine the size of the date or time item in number of digits. Integer-3 or integer-4 must be specified if the size of the date or time item cannot be determined at compile time. For a date or time item, the values of both integer-3 and integer-4 must be equal to or greater than 4.

mnemonic-name-1, mnemonic-name-2
For more information about mnemonic-name-1 or mnemonic-name2, refer to the description in "LOCALE Phrase" on page 160 and "LOCALE Phrase" on page 92.

USAGE For a Class Date-Time Item

If no USAGE clause is specified for an item of class date-time, USAGE DISPLAY is assumed. A USAGE of DISPLAY or PACKED-DECIMAL (COMP-3) can be explicitly specified for a date-time item. A USAGE of PACKED-DECIMAL can be specified for an item of class date-time, if literal-2 contains only conversion specifiers, and those specifiers will result in numeric digits.

FORMAT Clause and PICTURE CLAUSE Similarities

A FORMAT clause defines an implicit PICTURE clause. Although there is no PICTURE character-string that can easily describe a date or time item, for some formats, an approximate definition does exist. For example, a date item with a FORMAT '%y,%m,%d' is similar to the PICTURE 99/99/99, where the '/' PICTURE symbol is replaced with a ','.

LOCALE Phrase

The LOCALE phrase specifies the culturally-appropriate format of the date, time, or timestamp item.

When the LOCALE phrase is specified, without literal-2, the format and separator used for the date and time items is completely based on the locale.

When the LOCALE phrase is specified with literal-2, literal-2 determines the format of the item, but the conversion specifications are replaced with items based on the locale.

mnemonic-name-1, mnemonic-name-2
If a mnemonic-name is specified, the locale used for the date or time item is one associated with the mnemonic-name in the LOCALE clause of the SPECIAL-NAMES paragraph. If a mnemonic-name is not specified, the current locale is used. For more information about how to determine the current locale, refer to the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

LOCALE OF Special Register

The LOCALE OF special register returns the equivalent of a locale mnemonic-name associated with the specified data item. If the data item does not have a locale associated with it, the keyword COBOL is returned. The LOCALE OF special register cannot be modified, and can only be specified in the PROCEDURE DIVISION where a locale mnemonic-name is allowed.

A date-time data item can be used in expressions using the LOCALE OF special register.

DDS Data Types and FORMAT Literal Equivalent

In DDS, a date data type's format is specified with the DATFMT parameter. Valid DATFMT parameters include *MDY and *ISO. Along with the DATFMT keyword, a DATSEP keyword specifies a one-character value to be used as a separator between the month, day, and year value. A complete list of DATFMT parameters and their allowed DDS Date separators, along with their COBOL format literal equivalents, is shown in Table 10 on page 161.

In DDS, a time data type's format is specified with the TIMFMT parameter. Valid TIMFMT parameters include *HMS and *ISO. Along with the TIMFMT keyword, a TIMSEP keyword specifies a one-character value to be used as a separator between the hours, minutes, and second value. A complete list of TIMFMT parameters and their allowed DDS Time separators, along with their COBOL format literal equivalents, is shown in Table 11 on page 161.

Table 10. DDS Date Data Types and Their Equivalent ILE COBOL Format

IBM i Format	COBOL-Generated Format	Description	Format	Valid Separators	Length
*MDY	%m/%d/%y	Month/Day/Year	mm/dd/yy	/-.,space	8
*DMY	%d/%m/%y	Day/Month/Year	dd/mm/yy	/-.,space	8
*YMD	%y/%m/%d	Year/Month/Day	yy/mm/dd	/-.,space	8
*JUL	%y/%j	Julian	yy/ddd	/-.,space	6
*ISO	@Y-%m-%d	International Standards Organization	yyyy-mm-dd	-	10
*USA	%m/%d/@Y	IBM USA Standard	mm/dd/yyyy	/	10
*EUR	%d.%m.@Y	IBM European Standard	dd.mm.yyyy	.	10
*JIS	@Y-%m-%d	Japanese Industrial Standard Christian Era	yyyy-mm-dd	-	10

Table 11. DDS Time Data Types and Their Equivalent ILE COBOL Format

IBM i Format	COBOL-Generated Format	Description	Format	Valid Separators	Length
*HMS	%H:%M:%S	Hours:Minutes:Seconds	hh:mm:ss	:.,space	8
*ISO	%H.%M.%S	International Standards Organization	hh.mm.ss	.	8
*USA	%I:%M @p	IBM USA Standard. AM and PM can be any mix of upper and lower case.	hh:mm AM or hh:mm PM	:	8

Table 11. DDS Time Data Types and Their Equivalent ILE COBOL Format (continued)					
IBM i Format	**COBOL-Generated Format**	**Description**	**Format**	**Valid Separators**	**Length**
*EUR	%H.%M.%S	IBM European Standard	hh.mm.ss	.	8
*JIS	%H:%M:%S	Japanese Industrial Standard Christian Era	hh:mm:ss	:	8

FORMAT OF Special Register

The FORMAT OF phrase of the PROCEDURE DIVISION creates an implicit special register, called the FORMAT OF special register, whose contents equal the FORMAT literal of the data item referenced by the identifier. The FORMAT OF special register can only be specified for data items of class date-time. The length of this special register depends on the literal or locale specified in the FORMAT phrase for the data item.

The FORMAT OF special register has the implicit definition:

```
USAGE DISPLAY, PICTURE X(n)
where n equals the number of bytes of the implicit or explicit
FORMAT literal.
```

For example, consider the following data description entry for date data item `date2`:

```
05 date2 FORMAT DATE IS '%d,%m,%y'.
```

The following MOVE statement uses the intrinsic function CONVERT-DATE-TIME to convert date data item `date3` into the format of date data item `date2`. The FORMAT OF phrase creates an implicit special register whose content would be `%d,%m,%y`.

```
MOVE FUNCTION CONVERT-DATE-TIME(date3, DATE, FORMAT OF date2)
          TO alpha-num-date.
```

The length of the special register in this example is 8.

The following rules apply:

- The FORMAT OF special register cannot be modified, and can only be specified in the PROCEDURE DIVISION, where a FORMAT non-numeric literal is allowed.
- A separate FORMAT OF special register exists for each identifier referenced with the FORMAT OF phrase

GLOBAL Clause

The GLOBAL clause specifies that a data-name or constant-name is available to the program that declares it and to every program contained within the program that declares it, as long as the contained program does not itself have a declaration for that name. All data-names subordinate to, or condition-names or index-names associated with a global name, are global names.

GLOBAL Clause - Format

A data-name is global if the GLOBAL clause is specified either in the data description entry by which the data-name is declared or in another entry to which that data description entry is subordinate.

[IBM Extension] The GLOBAL clause can be specified in the Linkage and Local-Storage Sections, but only in data description entries whose level-number is 01. [End of IBM Extension]

In the same Data Division, the data description entries for any two data items for which the same data-name is specified must not include the GLOBAL clause.

A statement in a program contained directly or indirectly within a program which describes a global name can reference that name without describing it again.

[IBM Extension] If the TYPEDEF clause is specified with the GLOBAL clause, the scope of the GLOBAL clause applies to the type-name, and to any data items subordinate to the type-name. The GLOBAL attribute is not acquired by a data item that is defines using a global type-name within a TYPE clause. [End of IBM Extension]

Sharing Data

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

1. The data content of an external data record can be referenced from any program provided that program has described that data record.
2. If a program is contained within another program, both programs can refer to data possessing the global attribute either in the containing program or in any program that directly or indirectly contains the containing program.
3. A parameter passed by reference can be shared between the calling program and the called program.

JUSTIFIED Clause

The JUSTIFIED clause overrides standard positioning rules for a receiving item of the alphabetic or alphanumeric categories.

JUSTIFIED Clause - Format

Notes:

[1] Syntax-checked only

The JUSTIFIED clause may be specified only at the elementary level. RIGHT is an optional word that is syntax checked only and has no effect on the execution of the program.

The JUSTIFIED clause cannot be specified for numeric or numeric-edited items.

[IBM Extension] It can be specified for DBCS, DBCS-edited, and national items. [End of IBM Extension]

The JUSTIFIED clause is not allowed for:

- Level-66 (RENAMES) entries
- Level-88 (condition-name) entries
- Items described with the USAGE IS INDEX clause

[IBM Extension]

- Items described with the USAGE IS POINTER clause
- Items described with the USAGE IS PROCEDURE-POINTER clause
- External or internal floating-point items
- Items with the TYPE clause.

[End of IBM Extension]

[IBM Extension] The JUSTIFIED clause can be specified for an alphanumeric edited item. [End of IBM Extension]

When the JUSTIFIED clause is omitted, the rules for standard alignment are followed (see “Alignment Rules” on page 131).

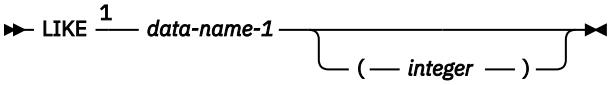
When the JUSTIFIED clause is specified for a receiving item, the data is aligned at the rightmost character position in the receiving item. Also:

- 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.

The JUSTIFIED clause does not affect initial values, as determined by the VALUE clause.

[IBM Extension] LIKE Clause

The LIKE clause allows you to define the PICTURE, USAGE, SIGN, and FORMAT 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.



LIKE Clause - Format

Notes:

[1] IBM Extension

data-name-1

Can refer to an elementary item, a group item, an index-name, or a type-name. The item referred to by data-name-1 is known as the *object* of the LIKE clause.

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 integer option to:

- Change the length of an edited item
- Change the length of an index, pointer, or procedure-pointer item
- Change the number of decimal places in a data item
- Change the length of an internal or external floating-point data item
- Change the length of a date, time, or timestamp 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, BLANK WHEN ZERO, and FORMAT 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 FORMAT characteristics that can be inherited include:

- The category of the item: date, time, or timestamp
- A FORMAT literal
- A SIZE phrase and LOCALE phrase.

For more information about the FORMAT clause, refer to "FORMAT Clause" on page 158.

[End of IBM Extension]

Comments Generated Based on Inherited USAGE Characteristics

The different USAGE clauses that you can specify for the original item result in a limited number of comments. Table 12 on page 165 illustrates this.

Table 12. Comments Generated based on Inherited USAGE Characteristics

Inherited USAGE Clause	Generated Comment
PACKED-DECIMAL COMPUTATIONAL COMPUTATIONAL-3	* USAGE IS PACKED-DECIMAL
COMP-1 COMUTATIONAL-1	* USAGE IS COMPUTATIONAL-1
COMP-2 COMUTATIONAL-2	* USAGE IS COMPUTATIONAL-2
BINARY COMP-4 COMPUTATIONAL-4	* USAGE IS BINARY
COMP-5 COMPUTATIONAL-5	* USAGE COMP-5
INDEX	*USAGE IS INDEX
NATIONAL	*USAGE IS NATIONAL
DISPLAY	This is the default usage, so a comment is not generated.
DISPLAY-1	* USAGE IS DISPLAY-1
POINTER	* USAGE IS POINTER
PROCEDURE-POINTER	* USAGE IS PROCEDURE-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
- FORMAT
- TYPE
- TYPEDEF.

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, and the LIKE clause has length modification, 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.

The object of a LIKE clause cannot contain the TYPE clause in its data description. If the object of a LIKE clause is a group item, then none of the items subordinate to this group can be defined using the TYPE clause. If the object of a LIKE clause is subordinate to a (level-01) group item, and an item which is subordinate to the level-01 group item contains a TYPE clause, then the type-name referenced in the TYPE clause must be fully defined at the point in the DATA DIVISION when the LIKE clause is used.

Coding 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)
```

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 201.) Format 1 of the OCCURS clause handles fixed-length tables. Format 2 of the OCCURS clause handles variable-length tables.

Table Handling Concepts

A table is a set of logically consecutive items, each of which has the same data description as the other items in the set. COBOL provides a method of data reference 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 occurrence 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.

The data-name of the data item containing the OCCURS clause is known as the **subject** of the OCCURS clause. When the subject of an OCCURS clause (or any data-item subordinate to it) is referenced, it must be subscripted or indexed unless:

- The subject of the OCCURS clause is used as the subject of the SEARCH statement.
- The subject (or subordinate data item) is the object of the ASCENDING/DESCENDING KEY clause.
- The subordinate data item is the object of the REDEFINES clause.

When the subject of an OCCURS clause is subscripted or indexed, it represents one occurrence within the table. Otherwise, the subject represents the entire table.

[IBM Extension]

An item whose usage is POINTER or PROCEDURE-POINTER can contain an OCCURS clause, or be subordinate to an item declared with an OCCURS clause.

Tables containing pointer or procedure-pointer data items are subject to pointer alignment as defined under "Pointer Alignment" on page 210. 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.

A boolean, external or internal floating-point, date, time, or timestamp item can contain an OCCURS clause, or be subordinate to an item declared with an OCCURS clause.

[End of IBM Extension]

Limitations

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 16 711 568.
- Table elements, including subordinate elements, have a size limit of 16 711 568 bytes.
- The OCCURS clause cannot appear in a data description entry that:
  - Has a level-number of 01, 66, 77, or 88.
  - Describes a redefined data item. However, a redefined item can be subordinate to an item containing an OCCURS clause.

Defining Tables

The ILE COBOL 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 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 on page 168 shows the storage layout for TABLE-THREE.

ELEMENT–ONE Occurs Three Times	ELEMENT–TWO Occurs Three Times	ELEMENT–THREE Occurs Two Times	Byte Displacement
ELEMENT–ONE (1)	ELEMENT–TWO (1, 1)	ELEMENT–THREE (1, 1, 1)	0
		ELEMENT–THREE (1, 1, 2)	8
	ELEMENT–TWO (1, 2)	ELEMENT–THREE (1, 2, 1)	16
		ELEMENT–THREE (1, 2, 2)	24
	ELEMENT–TWO (1, 3)	ELEMENT–THREE (1, 3, 1)	32
		ELEMENT–THREE (1, 3, 2)	40
ELEMENT–ONE (2)	ELEMENT–TWO (2, 1)	ELEMENT–THREE (2, 1, 1)	48
		ELEMENT–THREE (2, 1, 2)	56
	ELEMENT–TWO (2, 2)	ELEMENT–THREE (2, 2, 1)	64
		ELEMENT–THREE (2, 2, 2)	72
	ELEMENT–TWO (2, 3)	ELEMENT–THREE (2, 3, 1)	80
		ELEMENT–THREE (2, 3, 2)	88
ELEMENT–ONE (3)	ELEMENT–TWO (3, 1)	ELEMENT–THREE (3, 1, 1)	96
		ELEMENT–THREE (3, 1, 2)	104
	ELEMENT–TWO (3, 2)	ELEMENT–THREE (3, 2, 1)	112
		ELEMENT–THREE (3, 2, 2)	120
	ELEMENT–TWO (3, 3)	ELEMENT–THREE (3, 3, 1)	128
		ELEMENT–THREE (3, 3, 2)	136
			144

Figure 7. Storage Layout for TABLE-THREE

Referencing Table Elements

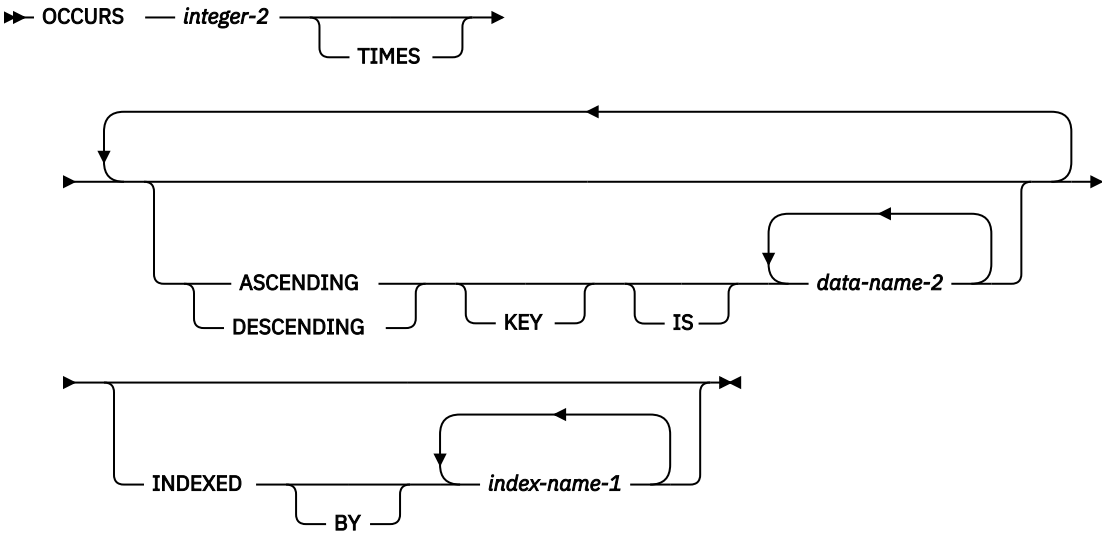
Whenever the user refers to a table element, or to any item associated with a table element, the reference must indicate which occurrence is intended.

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.

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.

OCCURS Clause - Format 1 - Fixed-Length Tables



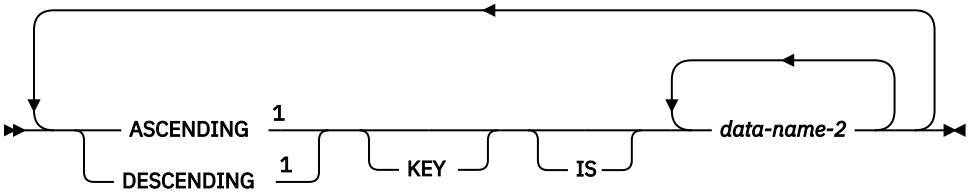
integer-2

Specifies the exact number of occurrences, and must be greater than zero.

In the ILE COBOL language, integer-2 must be between 1 and 16 711 568 bytes.

ASCENDING/DESCENDING KEY Phrase

Data is arranged in ascending or descending order (depending on the keyword specified) according to the values contained in data-name-2. The data-names are listed in their descending order of significance.



ASCENDING/DESCENDING KEY Phrase - Format

Notes:

[1] Cannot be used with boolean data type

The order is determined by the rules for comparison of operands (see “Relation Condition” on page 228). 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

ASCENDING/DESCENDING KEY Phrase Rules

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 can have DISPLAY, BINARY, PACKED-DECIMAL, or COMPUTATIONAL usage.

[IBM Extension]

- The KEY phrase can be specified in the OCCURS clause for a DBCS item.
- A key can have COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-3, COMPUTATIONAL-4, or COMPUTATIONAL-5 usage.
- A key can have DISPLAY-1 usage.
- A key can be an item of class date-time.

[End of IBM Extension]

ASCENDING/DESCENDING KEY Phrase Coding Example

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

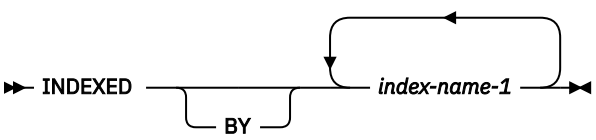
The INDEXED BY phrase specifies the indexes that can be used with this table. The INDEXED BY phrase is required if indexing is used to refer to this table element. See “Subscripting Using Index-Names (Indexing)” on page 51.

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

Indexes normally are allocated in static memory associated with the program containing the table. Consequently, indexes are in the last-used state when a program is re-entered. However, in the following cases, indexes are allocated on a per-invocation basis. Thus, you must SET the value of the index on every entry for indexes on tables in the following sections:

- Local-Storage Section.
- Linkage Section of a program compiled with the RECURSIVE attribute.

INDEXED BY Phrase - Format



index-name-1
Must follow the rules for formation of user-defined words. At least one character must be alphabetic.

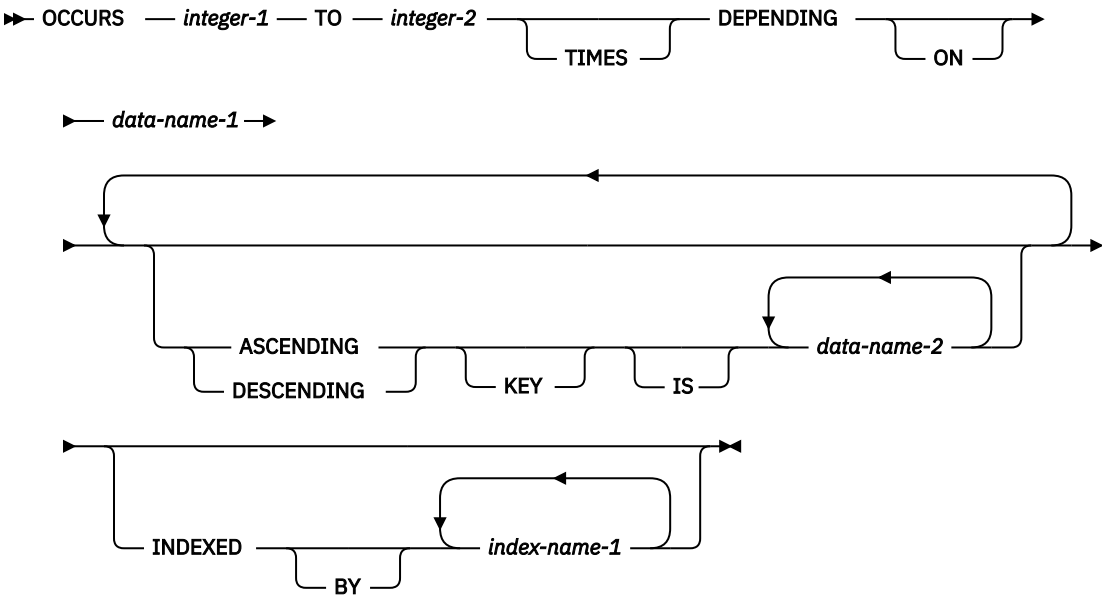
Each index-name specifies an index to be created by the compiler for use by the program. These index-names are not data-names, and are not identified elsewhere in the COBOL program; instead, they can be regarded as private special registers for the use of this object program only. As such, they are not data, or part of any data hierarchy; as such, each must be unique.

If a data item possessing the GLOBAL attribute includes a table accessed with an index, the index defined for the table also possess the GLOBAL attribute.

Variable-Length Tables

Variable-length tables are specified using Format 2 of the OCCURS clause.

OCCURS Clause - Format 2 - Variable-Length Tables



The **length** of the subject item is fixed; it is only the **number of repetitions** of the subject item that is variable.

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.

data-name-1
Specifies the **object** of the OCCURS DEPENDING ON clause; that is, the (integer) 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).

The object of the OCCURS DEPENDING ON clause may not be variably located; the object cannot follow an item that contains an OCCURS DEPENDING ON clause.

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 of the OCCURS DEPENDING ON clause must fall within the range integer-1 through

integer-2. This rule does not apply when the group being referenced is used in a CALL BY REFERENCE statement, provided that the group is not variably-located.

If the OCCURS clause is specified in a data description entry included in a record description entry containing the EXTERNAL clause, data-name-1 must reference a data item possessing the EXTERNAL attribute which is described in the same Data Division.

If the data description entry is subordinate to one containing the GLOBAL clause, data-name-1 must be a global name and must reference a data item which is described in the same Data Division.

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.

When reference modification is applied to a group item containing a variable-length table, reference modification creates a unique data item from the referenced data item. The length of this referenced data item is determined by first applying the previous rules. Subsequently, the rules for reference modification are applied to determine the length of the unique data item.

If a group item containing a variable-length table is used as an argument in the CALL statement USING phrase, the size of the storage for that parameter from the called program's point of view depends on how the argument is passed. If it is passed BY REFERENCE, the maximum size is described by the data description of the argument in the calling program. If it is passed BY CONTENT, the group item is considered as a sending item.

If the group item is followed by a non-subordinate item, the actual length (rather than the maximum length) will be used. At the time the subject of entry is referenced (or any data item subordinate or superordinate to the subject of entry is referenced), the object of the OCCURS DEPENDING ON clause must fall within the range integer-1 through integer-2.

The **subject** of an OCCURS clause is the data-name of the data item containing the OCCURS clause. The subject of an OCCURS clause may be subordinate to a type-name. 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, or by a level-66 item.

The OCCURS DEPENDING ON clause may not be specified as subordinate to another OCCURS clause.

[IBM Extension]

The following constitute complex OCCURS DEPENDING ON:

- Subordinate items can contain OCCURS DEPENDING ON clauses.
- Entries containing an OCCURS DEPENDING ON clause can be followed by non-subordinate items. Non-subordinate items, however, cannot be the object of an OCCURS DEPENDING ON clause.

- The location of any subordinate or non-subordinate item, following an item containing an OCCURS DEPENDING ON clause, is affected by the value of the OCCURS DEPENDING ON object.
- Entries subordinate to the subject of an OCCURS DEPENDING ON clause can contain OCCURS DEPENDING ON clauses.
- When implicit redefinition is used in a File Description (FD) entry, subordinate level items can contain OCCURS DEPENDING ON clauses.
- The INDEXED BY phrase can be specified for a table that has a subordinate item that contains an OCCURS DEPENDING ON clause.

For more information on complex OCCURS DEPENDING ON, see "Appendix H. Complex OCCURS DEPENDING ON" on page 585.

[End of IBM Extension]

All data-names used in the OCCURS clause may be qualified; they may not be subscripted or indexed.

The OCCURS or OCCURS DEPENDING ON clause cannot be specified in a data description entry that:

- Has a level number of 01, 66, 77, or 88.
- Describes a redefined data item. (However, a redefined item can be subordinate to an item containing an OCCURS clause.) See "REDEFINES Clause" on page 189.

The ASCENDING/DESCENDING KEY and INDEXED BY clauses are described under "Fixed-Length Tables" on page 168.

Note: If you use the OCCURS DEPENDING ON clause, the table must contain no more than 16 711 568 occurrences, the length of a table element must be no more than 16 711 568 bytes, and the length of the whole table must be no more than 16 711 568 bytes.

[IBM Extension]

Complex OCCURS DEPENDING ON is supported as an extension to the COBOL 85 Standard. The basic forms of complex ODO permitted by the compiler are:

- A data item described by an OCCURS clause with the DEPENDING ON option is followed by a non-subordinate element or group (variably-located item).
- A data item described by an OCCURS clause with the DEPENDING ON option is followed by a non-subordinate data item described by an OCCURS clause with the DEPENDING ON option (variably-located table).
- A data item described by an OCCURS clause with the DEPENDING ON option is nested within another data item described by an OCCURS clause with the DEPENDING ON option (table with variable-length elements).
- Index-name for a table with variable-length elements.

Complex ODO is tricky to use and can make maintaining your code more difficult. If you choose to use it in order to save disk space, follow the guidelines in "Appendix H. Complex OCCURS DEPENDING ON" on page 585.

[End of IBM Extension]

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.

Subscripting is related to the OCCURS clause through the number of dimensions in a table. For example, a 4-dimensional table will require four subscripts. You may think of subscripting as the COBOL way of identifying elements in a multidimensional array, which was defined through the OCCURS clause.

For detailed information, see "Subscripting" on page 49.

If the RANGE option is specified or implied, the system ensures 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.

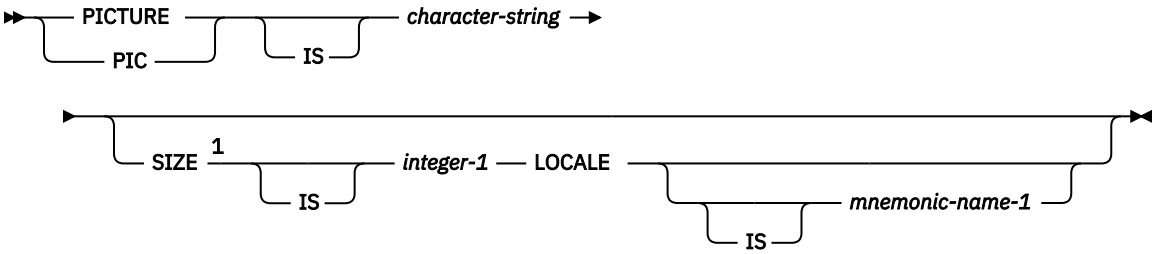
Note: See the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for comprehensive information on the CRTCBLMOD and CRTBNDCBL commands, and the PROCESS statement.

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.



PICTURE Clause - Format

Notes:

[1] IBM Extension

The PICTURE clause must be specified for every elementary item except an index data item, the subject of a LIKE, RENAMES, or TYPE clause.

The PICTURE character-string may contain a maximum of 90 characters. It consists of certain COBOL characters used as symbols. The allowable combinations determine the category of the elementary data item, except when the locale phrase is specified. A LOCALE phrase in a PICTURE clause defines a category numeric-edited item.

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.

A currency symbol is presented in a PICTURE character-string by either the dollar sign ($) or by a currency symbol specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph.

The currency symbol represents one or more character positions into which a currency-string is to be placed.

If multiple currency symbols are defined in the program, only one of the symbols may be repeated within the same PICTURE character-string.

When a single currency symbol appears in a picture string, it is a fixed insertion symbol. The size of the edited item will be increased by the number of characters contained in the corresponding currency-string.

When a string of two or more of the same currency symbols appear in a picture string, they are being used as floating insertion symbols. The size of the edited item will be increased by the number of characters contained in the corresponding currency-string for the first currency symbol present, and by a further character for each additional currency symbol in the PICTURE character-string.

The PICTURE clause is not allowed in:

- Descriptions of items described with USAGE IS INDEX

[IBM Extension]

- Internal floating-point (USAGE IS COMP-1 or USAGE IS COMP-2) data items
- USAGE IS POINTER or USAGE IS PROCEDURE-POINTER data items
- Date, time, or timestamp items
- Descriptions of items containing the TYPE clause.

[End of IBM Extension]

[IBM Extension] LOCALE Phrase

When the LOCALE phrase is specified in the PICTURE clause, editing is carried out according to the locale specifications. The following rules apply:

- A BLANK WHEN ZERO clause takes precedence over locale editing.
- When mnemonic-name-1 is specified, the locale used for editing and de-editing the item is the one associated with mnemonic-name-1 in the SPECIAL-NAMES paragraph. Otherwise the current locale is used.

  Note: Switching locales between the editing and de-editing stages of an item can cause unpredictable results. You must ensure that the locale used for editing an item is the same as the locale used for de-editing an item.
- If a currency sign symbol (cs) is specified in the picture string, the position, length, and character-string used for the currency sign are determined from the locale.
- The decimal separator, thousands separator, and grouping are determined by the locale.
- Decimal point alignment and zero replacement take place as described in “Alignment Rules” on page 131.
- If + is specified in the PICTURE character string, the way in which positive and negative numbers are represented is determined by a locale.
- The sending data is aligned on the decimal point, and (if necessary) truncated or padded with zeros at either end within the receiving character positions of the receiving data item. The data is also right-justified, with grouping and separators applied according to the locale specification. Leading zeros are replaced by blanks.

If, after formatting, the number of digit positions specified in the PICTURE character string do not fit into the receiving item, and there are excess digits in the sending item, digits are truncated on the left and an operating system escape message is issued.

[End of IBM Extension]

Symbols Used in the PICTURE Clause

The meaning of each PICTURE clause symbol is defined in the following tables:

- If the LOCALE phrase is *not* specified, see Table 13 on page 176
- If the LOCALE phrase *is* specified, see Table 14 on page 178

The sequence in which PICTURE clause symbols must be specified is shown in:

- Figure 8 on page 179, if the LOCALE phrase is not specified
- Figure 9 on page 180, if the LOCALE phrase is specified.

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.

If the OPTION parameter value *NOMONOPIC, or the PROCESS statement option NOMONOPIC is specified, the currency symbol used in the PICTURE character-string is case sensitive. That is, the lowercase letters corresponding to the uppercase letters for the PICTURE symbols A, B, C, D, E, G, N, P, R, S, V, X, and Z are equivalent to their uppercase representations in a PICTURE character-string. All other lowercase letters are not equivalent to their corresponding uppercase representations.

If the OPTION parameter value *MONOPIC, or the PROCESS statement option MONOPIC is specified, all alphabetic characters in a PICTURE character-string will be converted to uppercase (monocasing).

Table 13. PICTURE Clause Symbol Meanings When LOCALE Phrase NOT Specified

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 occupies 1 byte for non-DBCS data and 2 bytes for DBCS data.
E	[IBM Extension] Marks the start of the exponent in an external floating-point item. It occupies 1 byte of storage. [End of IBM Extension]
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 (63) 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.

Table 13. PICTURE Clause Symbol Meanings When LOCALE Phrase NOT Specified (continued)

Symbol	Meaning
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.
,	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.
G	[IBM Extension] A DBCS position, occupying two bytes of storage, counting as one character. It cannot be specified for a non DBCS item. USAGE must be explicitly defined as DISPLAY-1. [End of IBM Extension]

Table 13. PICTURE Clause Symbol Meanings When LOCALE Phrase NOT Specified (continued)	
Symbol	**Meaning**
N	[IBM Extension] • If usage is explicitly defined as NATIONAL, a national (UCS-2 or Unicode) character position. • If usage is explicitly defined as DISPLAY-1, a DBCS position that occupies two bytes of storage counting as one character. • If the USAGE clause is not specified for an elementary item, or for any group to which the data item belongs, the following rules apply: – If the NATIONAL compiler option is in effect, USAGE NATIONAL is implied. – Otherwise, USAGE DISPLAY-1 is implied. [End of IBM Extension]

Table 14. PICTURE Clause Symbol Meanings When LOCALE Phrase IS Specified	
Symbol	**Meaning**
9	A character position that contains a numeral and is counted in the number of numerals that may appear in the edited item.
.	An editing symbol that represents the decimal point for alignment purposes. 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. The decimal point character used at runtime is taken from the locale. **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.
+	Editing sign control symbol. The + indicates that the edited item is to be signed in accordance with the specified locale. If + is not specified, the edited item will be unsigned.
cs	The currency symbol in the character string indicates that the edited item is to include the currency string associated with the specified locale.

Figure 8 on page 179 shows the sequence in which PICTURE clause symbols must be specified if the LOCALE phrase *is not* specified. See the notes at the end of the figure. Figure 9 on page 180 shows the sequence in which PICTURE clause symbols must be specified if the LOCALE phrase *is* specified.

First Symbol / Second Symbol		Non-floating Insertion Symbols										Floating Insertion Symbols						Other Symbols								
		B	0	/	,	.	+ -	+ -	CR DB	$	E	Z *	Z *	+ -	+ -	$	$	9	A X	S	V	P	P	1	G	N
Non-floating Insertion Symbols	B	X	X	X	X	X	X			X		X	X	X	X	X	X	X	X		X		X		X	
	0	X	X	X	X	X	X			X		X	X	X	X	X	X	X	X		X		X			
	/	X	X	X	X	X	X			X		X	X	X	X	X	X	X	X		X		X			
	,	X	X	X	X	X	X			X		X	X	X	X	X	X	X			X		X			
	.	X	X	X	X		X			X		X		X		X		X								
	+ -																									
	+ -	X	X	X	X	X				X	X	X	X			X	X	X			X	X	X			
	CR DB	X	X	X	X	X				X		X	X			X	X	X			X	X	X			
	$						X																			
	E				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																									
	G	X																							X	
	N																									X

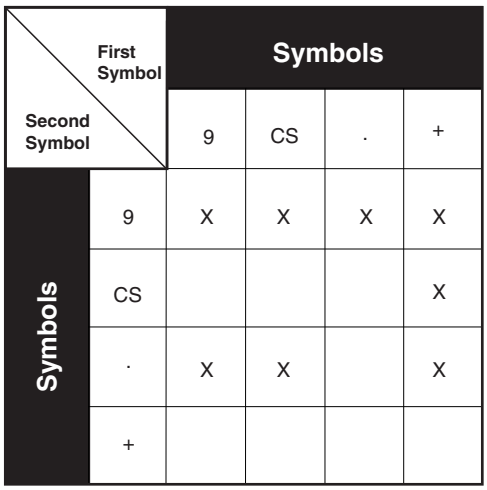
Figure 8. PICTURE Clause Symbol Sequence When LOCALE Phrase NOT Specified

Notes to Figure 8 on page 179:

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. [IBM Extension] The symbols G or N can appear alone in the PICTURE character-string. [End of IBM Extension]
5. 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. ({ }) indicate items that are mutually exclusive.
6. Braces ({}) indicate items that are mutually exclusive.

[IBM Extension]



First Symbol / Second Symbol		Symbols			
		9	CS	.	+
Symbols	9	X	X	X	X
	CS				X
	.	X	X		X
	+				

Figure 9. PICTURE Clause Symbol Sequence When LOCALE Phrase Specified

[End of IBM Extension]

Character-String Representation

The following symbols may appear more than once in one PICTURE character-string:

```
A  B  P  X  Z  9  0  /  ,  +  -  *  $
```

```
 [IBM Extension]  G      N [End of IBM Extension]
```

The following symbols may appear only once in one PICTURE character-string:

```
S V . CR DB
```

```
 [IBM Extension]  E  1 [End of IBM Extension]
```

If the LOCALE phrase is specified, only the symbol 9 can appear more than once. If the LOCALE phrase is specified, the following symbols may appear only once in one PICTURE character string:

```
. + cs
```

An integer enclosed in parentheses immediately following any of the symbols that may occur more than once in a PICTURE character string specifies the number of consecutive occurrences of that symbol. The integer may be specified by a constant name. The number of consecutive occurrences cannot exceed 16 711 568.

For example, the following two PICTURE clause specifications are equivalent:

```
PICTURE IS $99999.99CR
PICTURE IS $9(5).9(2)CR
```

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
- Boolean items

[IBM Extension]

- DBCS items
- DBCS-edited items
- National items
- External floating-point items

[End of IBM Extension]

Note: If the LOCALE phrase is specified in a PICTURE clause, the category of data defined by that PICTURE clause is numeric-edited only.

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).
  - National 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.
- [IBM Extension] For packed and zoned decimal numeric items, the number of digit positions can range from 1 through 63, inclusive. [End of IBM Extension]
- 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, COMPUTATIONAL-4, COMPUTATIONAL-5, or NATIONAL.

  For signed numeric items described with usage NATIONAL, the SIGN IS SEPARATE clause must be specified or implied.

[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 179), and the editing rules (see "PICTURE Clause Editing" on page 185). 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.

  – [IBM Extension] The number of digit positions represented in the character-string must be in the range 1 through 63, inclusive. [End of IBM Extension]

  – 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.

[IBM Extension] If the LOCALE Phrase is Specified

- The PICTURE character-string can contain the following symbols:

  ```
  9 . + cs (currency symbol for the locale)
  ```

  and of those symbols the following can only be used once:

  ```
  . + cs
  ```

- The number of character positions that can be used is specified in integer-1 in the SIZE phrase.
- 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.
- If the receiving item is a numeric-edited data item and the LOCALE phrase of the PICTURE clause is specified in its data description entry, the data is aligned as described in "LOCALE Phrase" on page 175.

[End of IBM Extension]

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 *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information about indicators.)

[End of IBM Extension]

[IBM Extension] DBCS Items

1. The PICTURE character-string can contain the symbol(s) G or N.
2. Each G or N represents a single DBCS character position (2 bytes).
3. When PICTURE clause symbol G is used, USAGE DISPLAY-1 must be specified.
4. When PICTURE clause symbol N is used, USAGE DISPLAY-1 must be implicitly or explicitly specified.
5. Associated VALUE clauses must specify a DBCS literal or the figurative constant SPACE/SPACES.

[End of IBM Extension]

[IBM Extension] DBCS-Edited Items

1. The PICTURE character-string is a combination of G's and B's with at least one of each.
2. Each G, and B represents a single DBCS character position (2 bytes).
3. USAGE DISPLAY-1 must be specified.
4. Associated VALUE clauses must specify a DBCS literal or the figurative constant SPACE/SPACES.

[End of IBM Extension]

[IBM Extension] National Items

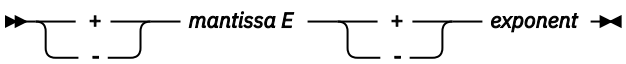
1. The PICTURE character-string can contain the symbol(s) N.
2. The contents of the item must be UCS-2 characters and must not be characters requiring multiple encoding units.
3. Each N represents a single UCS-2 character.
4. When PICTURE clause symbol N is used, USAGE NATIONAL must be implicitly or explicitly specified.
5. Associated VALUE clauses must specify an alphanumeric literal, a non-numeric literal, a national literal, or one of the following figurative constants:
   - SPACE/SPACES
   - ZERO/ZEROS/ZEROES
   - ALL *alphanumeric-literal*
   - ALL *national-literal*

[End of IBM Extension]

[IBM Extension] External Floating-Point Items

- The PICTURE string must have the following form:

  Format

  + OR -
  A sign character must immediately precede both the mantissa and the exponent. A + sign indicates that a positive sign is used in the output to represent positive values and that a negative sign represents negative values. A - sign indicates that a blank is used in the output to represent positive values and that a negative sign represents negative values. Each sign position occupies one byte of storage.

  mantissa
  The mantissa may contain the symbols:

  ```
  9 . V
  ```

  An actual decimal point is represented with a period (.) while an assumed decimal point is represented by a V. Either an actual or an assumed decimal point must be present in the mantissa; the decimal point can be leading, embedded, or trailing. The mantissa can contain from 1 to 16 numeric characters. The `.` and V are not included in the count of numeric characters.

  E
  Is used to indicate the separation of the mantissa and the exponent. It is required.

  exponent
  The exponent must consist of the symbols 99 or 999.

- The OCCURS, REDEFINES, LIKE, RENAMES, and TYPEDEF clauses can be associated with external floating-point items.
- The SIGN clause is accepted as documentation and has no effect on the representation of the sign.

- The SYNCHRONIZED clause is treated as documentation.
- The following clauses are not valid with external floating-point items:
  - BLANK WHEN ZERO
  - JUSTIFIED
  - VALUE

[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:

Table 15. Valid Editing for Each Data Category

Category	**Type of Editing**
Alphabetic	None
[IBM Extension] Boolean [End of IBM Extension]	None
[IBM Extension] DBCS [End of IBM Extension]	None
[IBM Extension] National [End of IBM Extension]	None
[IBM Extension] DBCS-edited [End of IBM Extension]	Simple insertion
Numeric	None
Alphanumeric	None
Alphanumeric edited	Simple insertion
Numeric edited	All
[IBM Extension] External floating-point [End of IBM Extension]	Special insertion

Simple Insertion Editing

This type of editing is valid for numeric-edited, and alphanumeric-edited items.

[IBM Extension] This type of editing is valid for DBCS-edited items. [End of IBM Extension]

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.

Table 16. Simple Insertion Editing – Valid Insertion Symbols for Each Data Category	
Category	**Valid Insertion Symbols**
Alphabetic	None
[IBM Extension] Boolean [End of IBM Extension]	None
[IBM Extension] DBCS [End of IBM Extension]	None
[IBM Extension] National [End of IBM Extension]	None
[IBM Extension] DBCS-edited [End of IBM Extension]	B
Numeric	None
Alphanumeric	None
Alphanumeric edited	B 0 / - . , &
Numeric edited	B 0 / ,

Examples of simple insertion editing:

```
PICTURE          Value of Data      Edited Results
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
GGBBGG           D1D2D3D4           D1D2    D3D4
```

Special Insertion Editing

This type of editing is valid only for:

- Numeric-edited items
- [IBM Extension] External floating-point items. [End of IBM Extension]

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
+999.99E+99     12345               +123.45E+02
```

Fixed Insertion Editing

This type of editing is valid only for numeric-edited items. The following insertion symbols are used:

- Currency symbol, for example $
- + - 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.

The currency symbol represents the position at which a currency sign is to appear. A currency sign may be the currency symbol itself, or a currency-string one or more characters in length that is specified in the CURRENCY SIGN clause of the SPECIAL-NAMES paragraph. The size of the edited item will be increased by the number of characters contained in the corresponding currency-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
U999.99	-123.45	EUR123.45 1
-$999.99	-123.456	-$123.45
-u999.99	-123.456	-USD123.45 2
-$999.99	+123.456	$123.45
$9999.99CR	+123.45	$0123.45
$9999.99DB	-123.45	$0123.45DB

1
For a currency sign defined as: `CURRENCY SIGN IS "EUR" PICTURE SYMBOL "U"`

2
For a currency sign defined as: `CURRENCY SIGN IS "USD" PICTURE SYMBOL "u"`

Note: Beware of situations where sign truncation would lead to negative amounts being shown as credits.

Floating Insertion Editing

This type of editing is valid only for numeric-edited items. The following symbols are used:

- Currency symbol, for example $
- + -

Within one PICTURE character-string, these symbols are mutually exclusive as floating insertion symbols.

Floating insertion editing is specified by including two or more consecutive floating insertion symbols in the PICTURE character-string.

A currency symbol represents a currency sign, which may either be the currency symbol itself, or a currency-string one or more characters in length that is specified in the CURRENCY SIGN clause of the SPECIAL-NAMES paragraph. The size of the edited item will be increased by the number of characters contained in the corresponding currency-string for the first currency symbol present, and by a further character for each additional currency symbol in the PICTURE character-string.

If the floating insertion symbol represents a single character, the symbols are used to represent all character positions into which the corresponding character could be inserted. The leftmost floating insertion symbol in the character-string represents the leftmost limit at which the character can appear in

the data item. The rightmost floating insertion symbol represents the rightmost limit at which the character can appear.

If the floating insertion symbol represents a multiple-character currency-string, the symbols are used to represent all the positions into which the final character of the currency-string could be inserted. The leftmost floating insertion symbol in the character-string represents the leftmost limit at which the final character of the currency-string can appear in the data item. The rightmost floating insertion symbol represents the rightmost limit at which the final character of the currency-string 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. Floating insertion symbols at or to the right of this limit represent numeric character positions. They may be replaced by numeric data, starting with the leading nonzero numeric character.

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 sign insertion symbol (+ or -), or the currency sign, 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. Any unused positions to the left of the insertion symbol or currency sign 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
- The number of characters in the floating insertion symbol.

Examples of floating insertion editing:

```
   PICTURE               Value of Data        Edited Result
        $$$$.99                 .123              $.12
        $$$9.99                 .12              $0.12
   $,$$$,999.99              -1234.56          $1,234.56
   U,UUU,UU9.99-             -1234.56         EUR1,234.56-
   u,uuu,uu9.99               1234.56         USD1,234.56
   +,+++,999.99            -123456.789          -123,456.78
  $$,$$$,$$$.99CR         -1234567          $1,234,567.00CR
  ++,+++,+++.+++              0000.00
```

Note: Beware of situations where sign truncation would lead to negative amounts being shown as credits.

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 symbols within one PICTURE character-string:

```
   Z  *  +  -  Currency symbol (for example, $)
```

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.

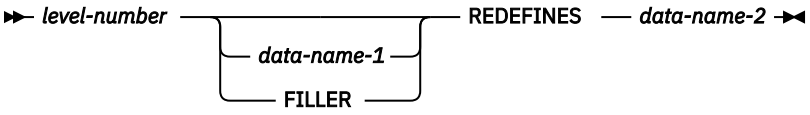
Examples of zero suppression and replacement editing:

```
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.

REDEFINES Clause - Format



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, and the data item being described is treated as though FILLER has been specified.

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/FILLER
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**. Contrast it with data-name-1, which is the REDEFINES clause **subject**.

[IBM Extension]

Both data-name-1 and data-name-2 can specify a pointer, procedure-pointer, external or internal floating-point data item, DBCS, national, date, time, or timestamp item.

[End of IBM Extension]

The following rules apply when coding the REDEFINES clause.

When more than one level-01 entry is written subordinate to an FD entry (and the level-01 entry is not a type-name), 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 and TYPE clause must not be specified. In addition, the TYPE clause must not be specified in any items subordinate to any of the level-01 entries.

Redefinition Process

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 containan 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. The original item, the redefined item, and all items subordinate to them cannot contain an OCCURS DEPENDING ON clause.

If the GLOBAL clause is used in the data description entry which contains the REDEFINES clause, only the subject of the clause possesses the global attribute. The EXTERNAL clause must not be specified on the same data description entry as a REDEFINES clause. If the object is GLOBAL or EXTERNAL, the subject does not inherit the attribute.

Data-name-1, the redefining item, may be smaller than data-name-2, the redefined item. It may only be larger than the redefined item if the redefined item is specified with a level-number of 01 and is not declared to be an external data record.

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. The redefining entry cannot contain a TYPEDEF clause. The redefining and redefined entries, and any subordinate entries must not contain a TYPE clause.

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 changed within a redefined 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.

Coding 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

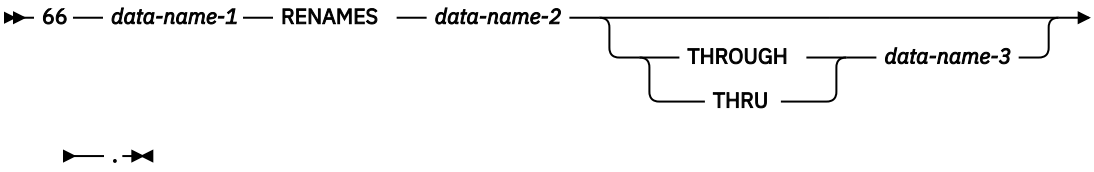
Undefined results may occur when:

- 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.

RENAMES Clause - Format



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.

[IBM Extension] Data-name-1 can specify a DBCS data item if data-name-2 specifies a DBCS data item and the THROUGH phrase is not specified. [End of IBM Extension]

[IBM Extension] Data-name-1 can specify a national data item if data-name-2 specifies a national data item and the THROUGH phrase is not specified. [End of IBM Extension]

[IBM Extension] If data-name-2 references one of the following data items, and the THROUGH phrase is not specified, data-name-1 can be one of the following types of data items:

- DBCS
- National
- Pointer or procedure-pointer
- Internal or external floating-point
- Date, time, or timestamp

[End of IBM Extension]

data-name-2, data-name-3
Identify the original grouping of elementary data items; that is, they must be 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 subordinate. In addition, the OCCURS DEPENDING ON clause must not be specified for any item defined between data-name-2 and data-name-3.

[IBM Extension] The TYPE clause must not be specified in the data descriptions of data-name-2, data-name-3, and items defined between data-name-2 and data-name-3, or any subordinates of these items. If data-name-2, data-name-3, or any items defined between data-name-2 and data-name-3 are subordinate to a group item defined using the TYPE clause, then data-name-1 must be subordinate to the same group item. [End of IBM Extension]

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 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 10 on page 194 illustrates valid and invalid RENAMES clause specifications.

Illustrations of Valid and Invalid RENAMES Clause Specifications

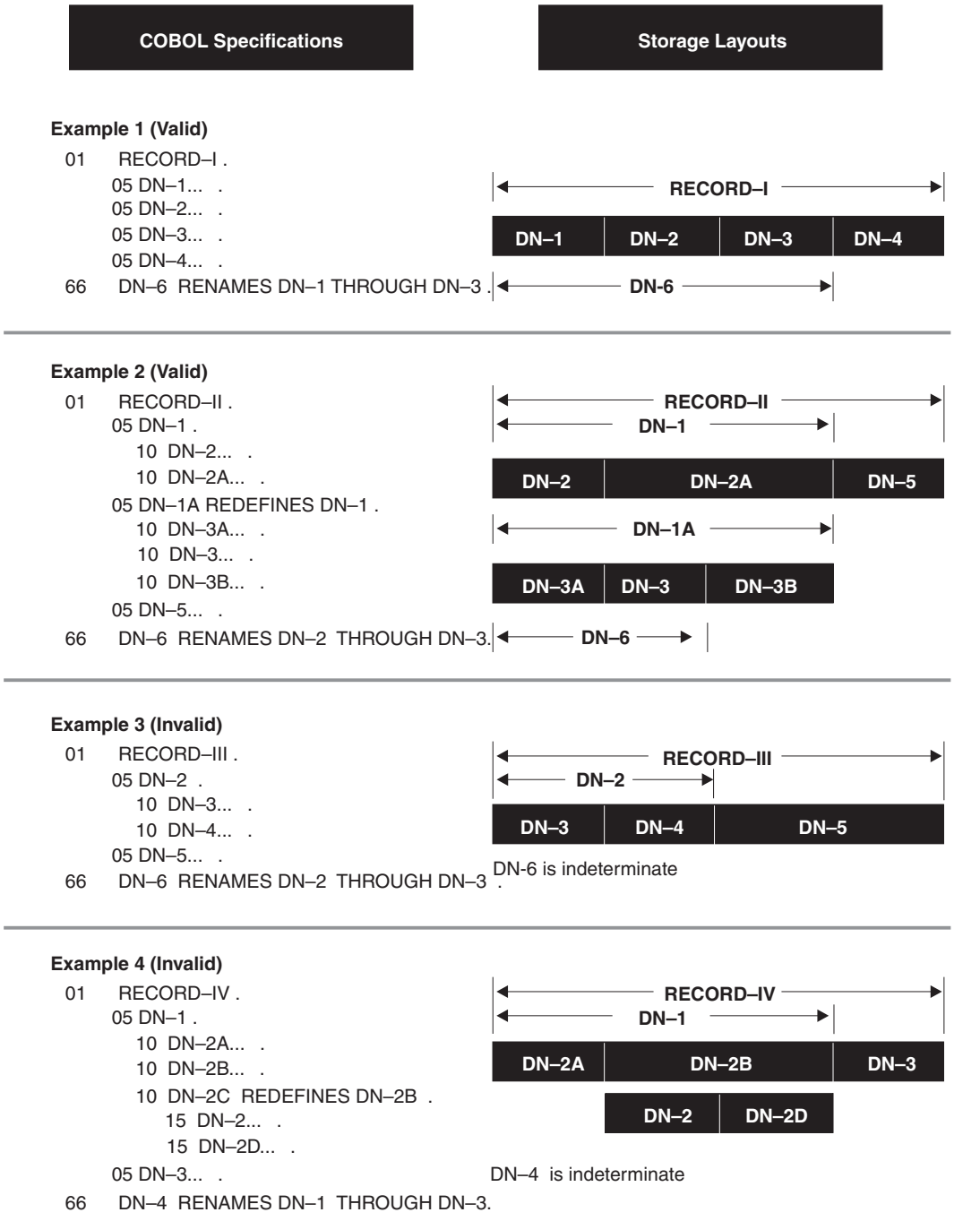
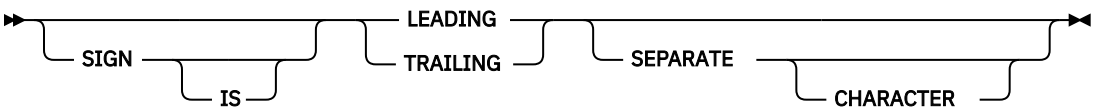


Figure 10. RENAMES Clause—Valid and Invalid Specifications

SIGN Clause

The SIGN clause specifies the position and mode of representation of the operational sign for a numeric entry.

SIGN Clause - 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 or USAGE IS NATIONAL must be specified, explicitly or implicitly.

When the SIGN clause is specified without the SEPARATE phrase, USAGE DISPLAY must be specified explicitly or implicitly. When SIGN IS SEPARATE is specified, either USAGE DISPLAY or USAGE NATIONAL can be specified.

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 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.

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.

[IBM Extension]

The SIGN clause is treated as documentation for external floating-point items. For internal floating-point items, the SIGN clause is invalid.

The SIGN clause cannot be specified if the FORMAT clause is specified.

The TYPE clause cannot be specified in the same data description entry as the SIGN clause.

[End of IBM Extension]

SEPARATE CHARACTER

If the SEPARATE CHARACTER phrase *is not* specified, then:

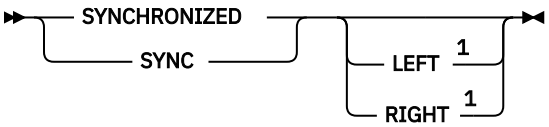
- The operational sign is presumed to be associated with the LEADING or TRAILING digit position, whichever is specified, of the elementary numeric data item. (In this instance, specification of SIGN IS TRAILING is the equivalent of the standard action of the compiler.)
- The character S in the PICTURE 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.

SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the alignment of an elementary item in storage. To use the SYNCHRONIZED clause, specify the *SYNC compiler option on the CRTCBLMOD or CRTBNDCBL command.



SYNCHRONIZED Clause - Format

Notes:

[1] Syntax-checked only.

When specified, the LEFT and the RIGHT phrases are syntax checked, but they have no effect on the execution of the program.

If synchronization is not specified, data is placed contiguously without filler space. If synchronization is specified, data is aligned along addresses which may be wholly divisible by 1, 2, 4, 8, or 16 bytes (where allowed - see Table 17 on page 197). This may require the (implicit) use of filler space, should the preceding data item not use all the bytes between boundaries.

Benefits of Synchronized Data

What is the benefit of synchronizing data? Improved performance in terms of its accessibility. The penalty is some wasted storage, due to increased record size (filler spaces become part of the record).

Level 01 items and pointers are aligned on 16-byte boundaries always, whether synchronization is specified or not. You are allowed to specify synchronization only for elementary items. It is not permitted for group items.

Figure 11 on page 196 illustrates the concept:

```
01      A
   05      A2  PIC  X(3)
   05      A3  PIC  9(5)   BINARY
01      B
   05      B2  PIC  X(16)
```

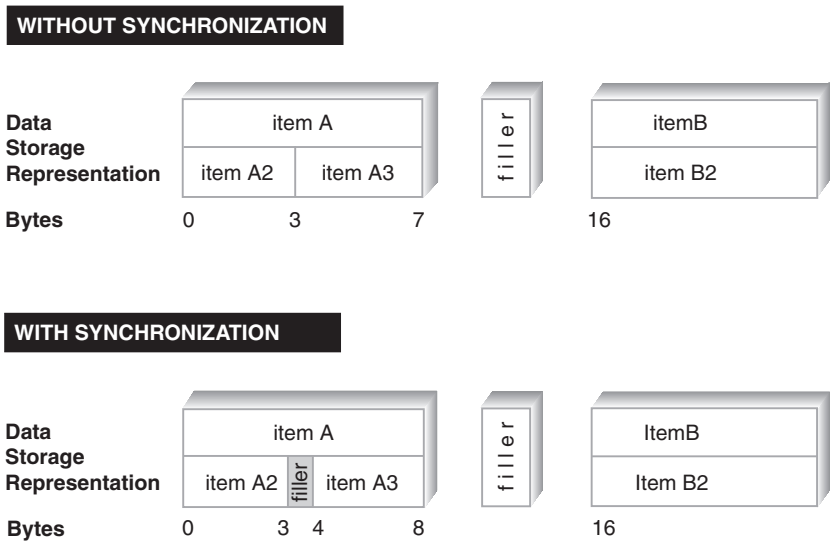


Figure 11. Data Storage Representation Without and With Synchronization

Figure 11 on page 196 shows that A and B are always aligned on 16 byte boundaries. Without synchronization, A2 and A3 are stored contiguously regardless of size. With synchronization, a 4 byte boundary is chosen (due to A3's type), and A3 is aligned accordingly. There is a one byte filler between A2 and A3. However, A3 should be accessed faster.

Synchronization and Offsets

In the preceding figure, note that A and B do not have to be following each other in actual storage. In other words, you cannot know if B starts 16 bytes after A's start, or 48 bytes (16 x 3), or 16 x N bytes. You must not attempt to retrieve synchronized data by specifying offsets.

[IBM Extension]

The SYNCHRONIZED clause is implicit for pointer data and procedure-pointer data items. Pointer data and procedure-pointer data items declared in the Linkage Section are not synchronized.

The SYNCHRONIZED clause cannot be specified in the same data description entry as the TYPE clause.

The SYNCHRONIZED clause is ignored for a DBCS, national, external floating-point, date, time, or timestamp data item.

The SYNCHRONIZED clause for a COMPUTATIONAL-1 data item aligns the data on a fullword boundary.

The SYNCHRONIZED clause for a COMPUTATIONAL-2 data item aligns the data on a doubleword boundary.

[End of IBM Extension]

Depending on the USAGE that is specified for an item, the SYNCHRONIZED clause has a particular effect. Table 17 on page 197 shows how the USAGE of an item determines the effect of the SYNCHRONIZED clause upon it.

Table 17. Data Item USAGE and the SYNCHRONIZED Clause

If the USAGE is...	**The SYNCHRONIZED clause...**
DISPLAY	is syntax checked but does not affect execution
DISPLAY-1 (DBCS)	is ignored
NATIONAL	is ignored
PACKED-DECIMAL	is syntax checked but does not affect execution
COMPUTATIONAL-1	aligns the data on a fullword boundary
COMPUTATIONAL-2	aligns the data on a doubleword boundary
COMPUTATIONAL-3	is syntax checked but does not affect execution
BINARY: PIC S9(1) through PIC S9(4)	aligns data item at a multiple of 2 relative to the beginning of the record
BINARY: PIC S9(5) through PIC S9(9)	aligns data item at a multiple of 4 relative to the beginning of the record
BINARY: PIC S9(10) through PIC S9(18)	aligns data item at a multiple of 8 relative to the beginning of the record
COMPUTATIONAL-4	functions the same as for USAGE BINARY
COMPUTATIONAL-5	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

Table 17. Data Item USAGE and the SYNCHRONIZED Clause (continued)	
If the USAGE is...	**The SYNCHRONIZED clause...**
PROCEDURE-POINTER	functions the same as for USAGE POINTER

The length of an elementary item is not affected by the SYNCHRONIZED clause.

Specifying the SYNCHRONIZED Clause with the OCCURS Clause

When the SYNCHRONIZED clause is specified for an item within the scope of the OCCURS clause, each occurrence of the item is synchronized.

Specifying the SYNCHRONIZED Clause with the REDEFINES 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. No padding characters are added for items containing the REDEFINES clause. 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.

FILLER Items

The 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, 8, 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 not 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 or *STDINZHEX00 options, 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 level 01 or 77 are aligned according to the following rules:

Area	Level Number	Boundary Alignment
Working-Storage Section	01 77	16 bytes 16 bytes
Local-Storage Section	01 77	16 bytes 16 bytes
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. Pointer data and procedure-pointer data items are not synchronized.

Example of Implicit FILLER

The following COBOL data description will produce the computer storage allocation shown in Figure 12 on page 199.

```
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).
```

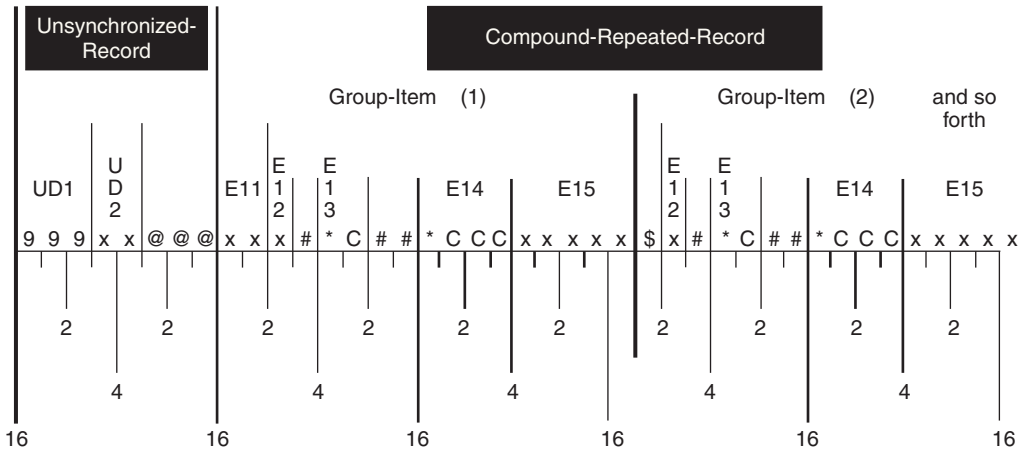


Figure 12. Computer Storage Allocation

@
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

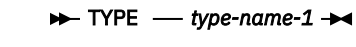
9
Indicates bytes allocated for a numeric DISPLAY character

X
Indicates bytes allocated for an alphanumeric DISPLAY character

C
Indicates bytes allocated for a BINARY data storage

[IBM Extension] TYPE Clause

The TYPE clause indicates that the data description of the subject of the entry is specified by a user-defined data type. The user-defined data type is defined using the TYPEDEF clause, which is described in "TYPEDEF Clause" on page 201.



The following general rules apply:

- If type-name-1 (defined using the TYPEDEF clause) describes a group item, then the subject of the TYPE clause is a group item whose subordinate elements have the same names, descriptions, and hierarchies as the subordinate elements of type-name-1.

  Note: Since the subject of the TYPE clause may have a level number as high as 49 and type-name-1 may be a group item with 49 levels, the number of levels of this hierarchy may exceed 49. In fact, since descriptions of type-names may reference other type-names, there is no limit to the number of levels in this hierarchy.
- If a VALUE clause is specified in the data description of the subject of the TYPE clause, any VALUE clause specified in the description of type-name-1 is ignored for this entry.
- The scoping rules for type names are similar to the scoping rules for data names.
- Reference modification is not allowed for an elementary item that is the subject of a TYPE clause.
- The description of type-name-1, including its subordinate data items, cannot contain a LIKE clause that references the subject of the TYPE clause (referencing type-name-1), or any group item to which the subject of the TYPE clause is subordinate.
- The description of type-name-1, including its subordinate data items, cannot contain a TYPE clause that references the record to which the subject of the TYPE clause (that references type-name-1), is subordinate

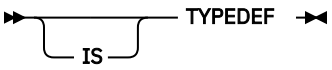
  For example, A is a group item defined using the TYPEDEF clause. B is also a group item defined using the TYPEDEF clause, but which also includes a subordinate item of TYPE A. This being the case, the type definition for A cannot include items of TYPE B.
- The subject of a TYPE clause cannot be renamed in whole, or in part.
- The subject of a TYPE clause cannot be redefined explicitly or implicitly.
- If the subject of a TYPE clause is subordinate to a group item, the data description of the group item cannot contain the USAGE clause.
- The TYPE clause cannot occur in a data description entry with the BLANK WHEN ZERO, FORMAT, JUSTIFIED, LIKE, PICTURE, REDEFINES, RENAMES, SIGN, SYNCHRONIZED, or USAGE clause.
- The TYPE clause can be specified in a data description entry with the EXTERNAL, GLOBAL, OCCURS, TYPEDEF, and VALUE clauses.

For more information about using the TYPE and TYPEDEF clauses, refer to the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

[End of IBM Extension]

[IBM Extension] TYPEDEF Clause

The TYPEDEF clause is used to create a new **user-defined data type**, type-name. The name of the new user-defined data type is the subject of the TYPEDEF clause. Data-name-1 must be specified with the TYPEDEF clause: FILLER cannot be used. The TYPEDEF clause must immediately follow data-name-1. After defining a new data type using the TYPEDEF clause, data items can be declared as this new data type using the TYPE clause. For more information about the TYPE clause, refer to "TYPE Clause" on page 200.



The TYPEDEF clause can only be specified for level 01 entries, which can also be group items. If a group item is specified, all subordinate items of the group become part of the type declaration. No storage is allocated for a type declaration.

The TYPEDEF clause cannot be specified in the same data description entry as the following clauses:

- EXTERNAL
- REDEFINES
- LIKE.

All of the other data description clauses, if they are specified, are assumed by any data item that is defined using the user-defined data type (within the TYPE clause).

TYPEDEF cannot be used with complex OCCURS DEPENDING ON. This means that you cannot specify an OCCURS DEPENDING ON clause within a table that is part of a TYPEDEF. For more information, see "Appendix H. Complex OCCURS DEPENDING ON" on page 585.

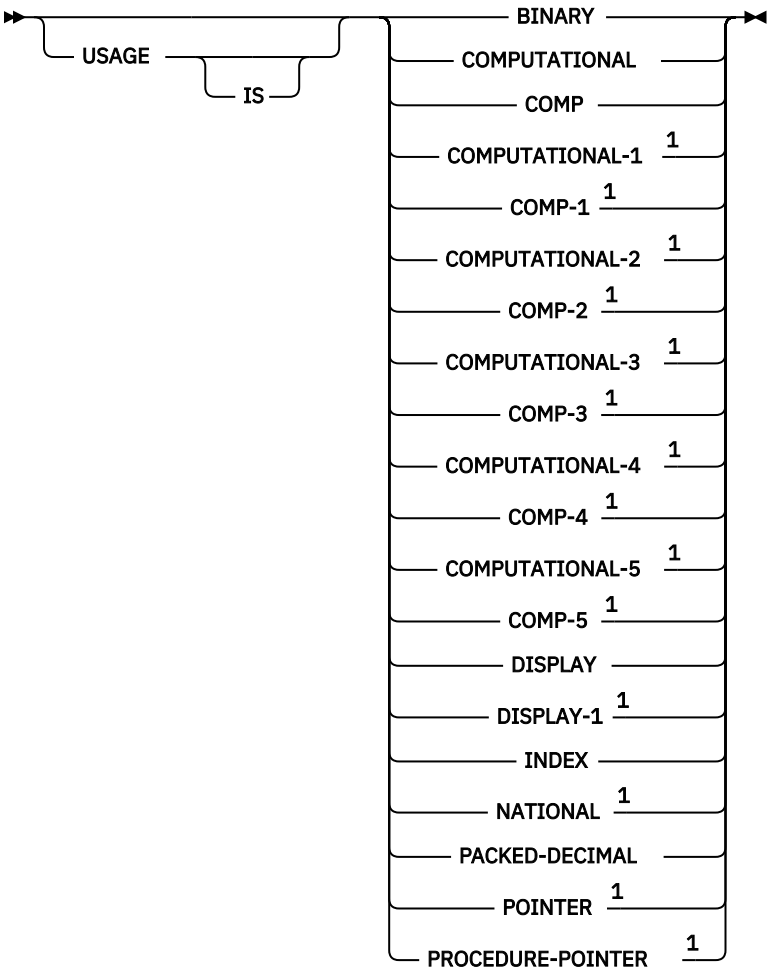
The TYPEDEF clause can only be specified in the WORKING-STORAGE, LOCAL-STORAGE, LINKAGE, or FILE sections of a program.

The TYPE clause can be specified in the same data description entry as the TYPEDEF clause.

[End of IBM Extension]

USAGE Clause

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.



USAGE Clause - Format

Notes:

[1] IBM Extension

The following table outlines the phrase that is used for the various data items specified by the USAGE clause.

Table 18. Usage Clause Data Items

Data Item	Phrase in USAGE Clause
Binary (computational item)	BINARY or COMPUTATIONAL-4[1] or COMP-4[1]
Native binary (computational item)	COMPUTATIONAL-5[1] or COMP-5[1]
Packed-decimal/Internal decimal (computational item)	PACKED-DECIMAL or COMPUTATIONAL or COMP or COMPUTATIONAL-3[1] or COMP-3[1]

Table 18. Usage Clause Data Items (continued)	
Data Item	**Phrase in USAGE Clause**
Internal floating point (computational item)	COMPUTATIONAL-1[1] or COMP-1[1] (4–byte) COMPUTATIONAL-2[1] or COMP-2[1] (8–byte)
Numeric DISPLAY items External decimal (zoned decimal) External floating-point[1]	DISPLAY
Non-numeric DISPLAY items Alphabetic Alphanumeric Alphanumeric-edited Numeric-edited items Boolean[1] Date time, and timestamp[1]	DISPLAY
DBCS[1] DBCS-edited[1]	DISPLAY-1
National[1]	NATIONAL
Index	INDEX
Pointer[1]	POINTER
Procedure-pointer[1]	PROCEDURE-POINTER
Note: 1. IBM Extension	

The USAGE clause can be specified for an entry at any level (other than 66 or 88). However, if it is specified at the group level, it applies to each elementary item in the group rather than to the group itself. The usage of an elementary item must not contradict the usage specified on the owning group item.

When the USAGE clause is not specified at either the group or elementary level, it is assumed that the usage is DISPLAY.

[IBM Extension]

The TYPE clause cannot be specified in the same data description entry as the USAGE clause.

Data description entries with a TYPE clause cannot be subordinate to a data description entry that contains a USAGE clause. For example, the following is illegal:

```
01 FLAGS   USAGE   DISPLAY.
   05 F-STATUS    TYPE CHAR.
   05 FLAG-ACTIVE TYPE CHAR.
```

[End of IBM Extension]

Computational Items

A computational item is a value used in arithmetic operations. Computational items must be numeric. These include binary, packed-decimal, and internal floating point data items.

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 245).

The maximum length of a computational item is 18 decimal digits.

[IBM Extension] The maximum length of a packed-decimal computational item is 31 decimal digits. [End of IBM Extension]

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.

[IBM Extension] Unlike other computational items, COMPUTATIONAL-1 and COMPUTATIONAL-2 items (internal floating-point) cannot have PICTURE strings. [End of IBM Extension]

BINARY Phrase

The BINARY phrase is specified for binary data items. Such items have a decimal equivalent consisting of the decimal digits 0 through 9, plus a sign.

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.

PACKED-DECIMAL Phrase

The PACKED-DECIMAL phrase is 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 13 on page 208.

[IBM Extension]

The maximum length of a packed-decimal computational item is 63 decimal digits.

PACKED-DECIMAL may also be specified for date and time items whose FORMAT literal contains only conversion specifiers. These conversion specifiers must only be able to contain numeric digits.

[End of IBM Extension]

COMPUTATIONAL or COMP Phrase

The COMPUTATIONAL or COMP phrase is 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. Of all USAGEs, USAGE COMP is the most efficient in terms of operational performance.

For the ILE COBOL compiler, the COMPUTATIONAL phrase is synonymous with

- USAGE COMP-4(Binary), if option COMPASBIN is specified
- Otherwise, PACKED-DECIMAL.

[IBM Extension] COMPUTATIONAL-1 or COMP-1 Phrase

The COMPUTATIONAL-1 or COMP-1 phrase is specified for internal floating-point items (single precision). COMP-1 items are 4 bytes long. The sign is contained in the first bit of the leftmost byte and the exponent is contained in the next 8 bits. The last 23 bits contain the mantissa. For conditional expressions, the class condition cannot be used for COMP-1 or COMPUTATIONAL-1 internal floating-point data items.

[End of IBM Extension]

[IBM Extension] COMPUTATIONAL-2 or COMP-2 Phrase

The COMPUTATIONAL-2 or COMP-2 phrase is specified for internal floating-point items (double precision). COMP-2 items are 8 bytes long. The sign is contained in the first bit of the leftmost byte and the next 11 bits contain the exponent. The remaining 52 bits contain the mantissa. For conditional expressions, the class condition cannot be used for COMPUTATIONAL-2 or COMP-2 internal floating-point data items.

[End of IBM Extension]

[IBM Extension] COMPUTATIONAL-3 or COMP-3 Phrase (Internal Decimal)

This is the equivalent of PACKED-DECIMAL.

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.

The contents of the leftmost (unused) digit position of the storage allocated for a packed decimal item that contains an even number of digits can change when the value of the item is changed. This might lead to unexpected results if, for example, the item is redefined, forms part of a group field, or is used as a key to an indexed file.

[End of IBM Extension]

[IBM Extension] COMPUTATIONAL-4 or COMP-4 Phrase (Binary)

This is the equivalent of BINARY.

[End of IBM Extension]

[IBM Extension] COMPUTATIONAL-5 or COMP-5 Phrase (Binary)

These data items are represented in storage as binary data. The data items can contain values up to the capacity of the native binary representation (2, 4 or 8 bytes), rather than being limited to the value implied by the number of nines in the picture for the item (as is the case for USAGE BINARY data). When numeric data is moved or stored into a COMP-5 item, truncation occurs at the binary field size rather than at the COBOL picture size limit. When a COMP-5 item is referenced, the full binary field size is used in the operation.

The *NOSTDTRUNC compiler option or the NOSTDTRUNC PROCESS option causes BINARY data items (USAGE BINARY, COMP-4) to be handled as if they were declared USAGE COMP-5. The only exception is

that unsigned BINARY data items always have a sign bit, so the maximum and minimum values for an unsigned BINARY data item are the same as for a signed BINARY data item.

The following table shows several picture character strings, the resulting storage representation, and the range of values for data items described with USAGE COMP-5.

Table 19. Storage Representation for COMP-5 Data Items

Picture	Storage representation	Numeric values
S9(1) through S9(4)	Binary halfword (2 bytes)	-32768 through +32767
S9(5) through S9(9)	Binary fullword (4 bytes)	-2,147,483,648 through +2,147,483,647
S9(10) through S9(18)	Binary doubleword (8 bytes)	-9,223,372,036,854,775,808 through +9,223,372,036,854,775,807
9(1) through 9(4)	Binary halfword (2 bytes)	0 through 65535
9(5) through 9(9)	Binary fullword (4 bytes)	0 through 4,294,967,295
9(10) through 9(18)	Binary doubleword (8 bytes)	0 through 18,446,744,073,709,551,615

The picture for a COMP-5 data item can specify a scaling factor (that is, decimal positions or implied integer positions). In this case, the maximal capacities listed in the table above must be scaled appropriately. For example, a data item described with PICTURE S99V99 COMP-5 is represented in storage as a binary halfword, and supports a range of values from -327.68 to +327.67.

Usage note: When the ON SIZE ERROR phrase is used on an arithmetic statement and a receiver is defined with USAGE COMP-5, the maximum value that the receiver can contain is the value implied by the item's decimal PICTURE character-string. Any attempt to store a value larger than this maximum will result in a size error condition.

[End of IBM Extension]

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
- External decimal (numeric)

[IBM Extension]

- Boolean
- Date, time, and timestamp
- External floating-point.

[End of IBM Extension]

[IBM Extension] For conditional expressions, the class condition cannot be used for external floating-point data items, which have a USAGE DISPLAY. [End of IBM Extension]

Alphabetic, alphanumeric, alphanumeric-edited, and numeric-edited items are discussed in “Data Categories and PICTURE Rules” on page 181.

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 (Numeric)

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.

[IBM Extension] The maximum length of an external decimal item is 63 digits. [End of IBM Extension]

[IBM Extension] External Floating Point (Numeric)

An **external floating point** item is a character string which has the following format:

mantissa sign
+ or - (mandatory)

mantissa
A numeric value between 1 and 16 digits in length that may in addition contain either a period (.) to denote an explicit decimal point, or a V to denote an implicit decimal point. The decimal point symbol may appear in any position of the mantissa.

E
A constant that introduces the exponent.

exponent sign
+ or - (mandatory)

exponent
A two- or three-digit numeric value.

[End of IBM Extension]

Internal representation of numeric items

Figure 13 on page 208 shows the internal representation of numeric items as specified by the USAGE clause. Numeric DISPLAY items include external decimal and external floating point data items. The computational numeric items are also shown in this figure: binary, internal decimal, and internal floating point data items.

ITEM	DESCRIPTION	VALUE	INTERNAL REPRESENTATION*
External Decimal	PIC S9999 DISPLAY	+1234 –1234 1234	F1 F2 F3 F4 F1 F2 F3 D4 F1 F2 F3 F4
	PIC 9999 DISPLAY	+1234 –1234 1234	F1 F2 F3 F4 F1 F2 F3 F4 F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN LEADING	+1234 –1234 1234	F1 F2 F3 F4 D1 F2 F3 F4 F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN TRAILING SEPARATE	+1234 –1234 1234	F1 F2 F3 F4 4E F1 F2 F3 F4 60 F1 F2 F3 F4 4E
	PIC S9999 DISPLAY SIGN LEADING SEPARATE	+1234 –1234 1234	4E F1 F2 F3 F4 60 F1 F2 F3 F4 4E F1 F2 F3 F4
Internal Decimal	PIC S9999 {COMP } {COMP–3}	+1234 –1234	01 23 4F 01 23 4D
	PIC 9999 {COMP } {COMP–3}	+1234 –1234	01 23 4F 01 23 4F
Binary	PIC S9999 COMP–4	+1234 –1234	04 D2 FB 2E
	PIC 9999 COMP–4	+1234 –1234	04 D2 04 D2
Internal Floating Point	COMP–1	+1234 –1234	44 9A 40 00 C4 9A 40 00
Internal Floating Point	COMP–2	+1234 –1234	40 93 48 00 00 00 00 00 C0 93 48 00 00 00 00 00
External Floating Point	PIC +9(2).9(2)E+99 DISPLAY	+1234 –1234	4E F1 F2 4B F3 F4 C5 4E F0 F2 60 F1 F2 4B F3 F4 C5 4E F0 F2

* 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 Specifications of SIGN TRAILING (without the SEPARATE CHARACTER option) is equivalent of the standard action of the compiler.

Note: The internal representation of native binary COMP-5 numeric items is the same as the internal representation of binary COMP-4 numeric items.

Figure 13. Internal Representation of Numeric Items

Table 20. Internal representation of numeric national items

Item	Description	Value	Internal representation
National decimal	PIC 9999 NATIONAL	1234	00 31 00 32 00 33 00 34
	PIC S9999 NATIONAL SIGN LEADING SEPARATE	+1234	00 2B 00 31 00 32 00 33 00 34
		-1234	00 2D 00 31 00 32 00 33 00 34
	PIC S9999 NATIONAL SIGN TRAILING SEPARATE	+1234	00 31 00 32 00 33 00 34 00 2B
		-1234	00 31 00 32 00 33 00 34 00 2D

[IBM Extension] DISPLAY-1 Phrase

The DISPLAY-1 phrase defines an item as DBCS or DBCS-edited.

[End of IBM Extension]

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
```

Any attempt to set an index-name to a value greater than 999 999 999 will leave the index-name value undefined.

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, TYPE, VALUE, or FORMAT clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

If a source program is to be portable to other systems, it must not depend on the content of the index data item when stored in external records (since the content is system specific).

[IBM Extension] NATIONAL Phrase

The NATIONAL phrase defines an item as national. The picture string of the corresponding data item can only contain one N or multple Ns.

[End of IBM Extension]

[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 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 and 7 only)
- A relation condition
- The USING phrase of a CALL statement or Procedure Division header
- Expressions involving ADDRESS OF or LENGTH OF.
- The argument on an intrinsic function

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, TYPE, BLANK WHEN ZERO, and FORMAT 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.

USAGE IS POINTER is implicitly specified for the ADDRESS OF special register.

You cannot treat ILE COBOL pointer data items as ordinary numbers.

[End of IBM Extension]

Pointer Alignment

For the purposes of this section on pointer alignment, the term pointer refers to both pointer data items and procedure-pointer data items.

When a pointer 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 as a pointer in the Working-Storage, Local-Storage or File sections is aligned. If the pointer 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 to make sure that it is also in alignment. The compiler issues a warning when it adds these FILLER items.

In the Linkage section:

- If the process option NOLSPTRALIGN is in effect, the compiler does not add FILLER items to the structure. The compiler issues warnings regarding its assumption that you have aligned the 01-level items.
- If the process option LSPTRALIGN is in effect, the data item described as pointer is also aligned.

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, Local-Storage or File sections, 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 or USAGE IS PROCEDURE-POINTER clause, but this clause is already implicit for pointers.

If the pointer is part of a table, the first item in the table is aligned, and to make sure that all occurrences of the pointer are also aligned, a filler item might be added to the end of the table.

To avoid adding FILLER items to data structures, place pointers at the beginning of the structures.

[IBM Extension] PROCEDURE-POINTER Phrase

A data item defined with the PROCEDURE-POINTER phrase is a **procedure-pointer data item**. It is a 16-byte elementary item containing the address of an entry point to an ILE procedure or program object (*PGM), such as:

- The entry point of the outermost ILE COBOL program (an ILE procedure) in the compilation unit defined by the PROGRAM-ID statement
- An entry point of a non-COBOL program, such as an ILE C function (an ILE procedure)
- An entry point of a program (*PGM).

A procedure-pointer data item may only be used in:

- The SET statement
- A relation condition
- The USING phrase of a CALL statement, or the Procedure Division header
- Expressions involving ADDRESS OF and LENGTH OF
- The CALL statement as a target
- The argument on an intrinsic function

Like pointer data items, procedure-pointer data items must be in alignment.

[End of IBM Extension]

Usage Rules

- The USAGE IS PROCEDURE-POINTER clause cannot be written at level-88.
- In a group item described with the USAGE IS PROCEDURE-POINTER clause, the elementary items within the group are procedure-pointer data items (the group itself is not a procedure-pointer).
- The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.
- Procedure-pointer data items can be part of a group that is referred to in a MOVE statement, or an input/output statement. However, there is no conversion of values when the statement is executed.
- A procedure-pointer data item can be written to a file, but if you later read the same record containing the procedure-pointer, the item will no longer represent a valid address.
- GLOBAL, EXTERNAL, OCCURS, SYNCHRONIZED, and LIKE clauses may be used with USAGE IS PROCEDURE-POINTER.
- A procedure-pointer may be the subject or object of a REDEFINES clause.
- A VALUE clause for a procedure-pointer data item can contain only NULL or NULLS.

- JUSTIFIED, PICTURE, TYPE, BLANK WHEN ZERO, and FORMAT clauses cannot describe group or elementary items defined with the USAGE IS PROCEDURE-POINTER clause.
- A procedure-pointer data item cannot be a conditional variable, does not belong to any class or category, and is ignored in CORRESPONDING operations.

VALUE Clause

The VALUE clause specifies the initial contents of a data item or the value(s) associated with a condition-name.

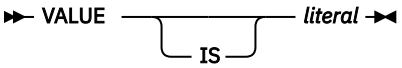
The use of the VALUE clause differs depending on the Data Division section in which it is specified.

[IBM Extension] In the Linkage section, a VALUE clause used in entries other than condition-names is treated as a comment. [End of IBM Extension]

In the File and Linkage sections, the VALUE clause must be used only in condition-name and type-name entries. In the Working-Storage and Local-Storage Sections, the VALUE clause may be used in condition-name entries, type-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.

VALUE Clause - Format 1 - Literal Value

VALUE Clause - 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. Each structure that contains the DEPENDING ON phrase of the OCCURS clause is assumed to contain the maximum number of occurrences for the purposes of VALUE initialization.

The VALUE clause must not be specified for a data description entry that contains, or is subordinate to, an entry containing an EXTERNAL clause or a REDEFINES clause. This rule does not apply to condition-name entries.

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 a USAGE (other than USAGE DISPLAY) clause.

The VALUE clause must not conflict with other clauses in the data description entry, or in the data description of this entry's hierarchy.

[IBM Extension]

Any VALUE clause associated with COMPUTATIONAL-1 or COMPUTATIONAL-2 (internal floating-point) items must specify a floating-point literal. The condition-name VALUE phrase must also specify a floating-point literal. In addition, the figurative constant ZERO and both integer and decimal forms of the zero literal can be specified in a floating-point VALUE clause or condition-name VALUE phrase.

For more information on floating-point literal values, see “Floating-Point Literals” on page 37.

A VALUE clause cannot be specified for external floating-point items.

A VALUE clause associated with a DBCS item must contain a DBCS literal or the figurative constant SPACE or SPACES.

A VALUE clause associated with a national character (PIC N) item must contain a non-numeric literal, a national literal, or the figurative constant SPACE or SPACES.

A VALUE clause associated with a national numeric (PIC 9) item must contain a numeric literal or the figurative constant ZERO/ZEROS/ZEROES.

A VALUE clause that specifies a national literal can be associated only with a data item of class national.

A VALUE clause that specifies a DBCS literal can be associated only with a data item of class DBCS.

A VALUE clause may be specified in the data description entry for a type-name. Such a VALUE clause is used to initialize any data name (which is not a type-name), that is defined using a TYPE clause that references such a type-name. If a VALUE clause is specified in the data description of the subject of a TYPE clause, any VALUE clause specified in the description of the associated type-name is ignored for this entry.

A data item cannot contain a VALUE clause if the prior data item contains an OCCURS clause with the DEPENDING ON phrase. A variably located item cannot contain the VALUE clause.

A VALUE clause associated with a date, time, or timestamp item must be a non-numeric literal. The literal is aligned according to alignment rules. No formatting of the literal is done to match conversion specifiers or LOCALE definition, except if the USAGE of the item is PACKED-DECIMAL, in which case the non-numeric literal is converted to packed.

[End of IBM Extension]

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).
- With an exception, 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.

  The exceptions are the following:

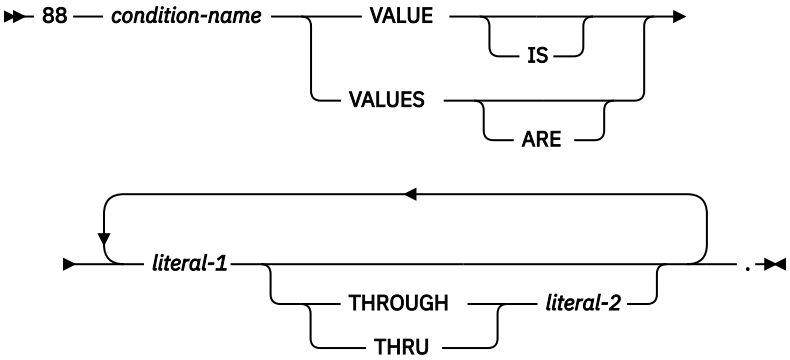
  – Data items described with usage COMP-5 that do not have a picture symbol P in their PICTURE clause.
  – When the *NOSTDTRUNC compiler option is in effect, data items described with usage BINARY or COMP-4 that do not have a picture symbol P in their PICTURE clause.

  A VALUE clause for these items can have a value up to the capacity of the native binary representation.
- 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.

VALUE Clause - Format 2 - Condition-Name Value

VALUE Clause - 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 225).

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.

[IBM Extension]

If the associated conditional variable is a DBCS data item, all the literals specified for the THROUGH phrase must be DBCS literals (or the figurative constants SPACE, SPACES). The range of DBCS literals specified for the THROUGH phrase is based on the binary collating sequence of the hexadecimal values of the DBCS characters.

If the associated conditional variable is a national data item, all the literals specified for the THROUGH phrase must be non-numeric literals, national literals (or the figurative constants SPACE, SPACES). The range of the literals specified for the THROUGH phrase is based on the binary collating sequence of the hexadecimal values of the national characters.

[End of IBM Extension]

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 condition-name entries associated with a particular conditional variable must immediately follow the conditional variable entry. The conditional variable can be any data description entry except:
  - A level-66 item (RENAMES clause)
  - A data item whose USAGE IS INDEX
  - An item whose USAGE IS POINTER or PROCEDURE-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.

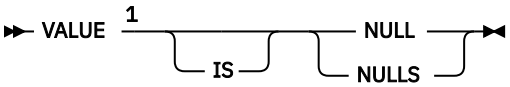
[IBM Extension]

- The VALUE clause is allowed for internal floating-point data items.
- The VALUE clause is allowed for DBCS items. Relation tests for DBCS data items are performed according to the rules for comparison of DBCS items.
- The VALUE clause is allowed for national items. Relation tests for national data items are performed according to the rules for comparison of national items.
- A condition-name can be associated with a date, time, or timestamp item. In this case:
  - The condition-name value must be specified as a non-numeric literal
  - Each condition-name implicitly has the FORMAT characteristics of the conditional variable. Thus, any relation test involving this condition-name is performed in accordance with the rules for comparing items of class date-time.
  - A THROUGH phrase can be specified when a conditional variable is of class date-time. In this case, the time or date of literal-1 must be less than literal-2.

[End of IBM Extension]

- 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.

[IBM Extension] VALUE Clause - Format 3 - NULL Value



VALUE Clause - Format 3 - NULL Value

Notes:

[1] IBM Extension

This format assigns an address that is not valid to a pointer data item or a procedure-pointer data item. 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 or USAGE IS PROCEDURE-POINTER.

[End of IBM Extension]

Chapter 7. Procedure Division

Procedure Division Overview

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.

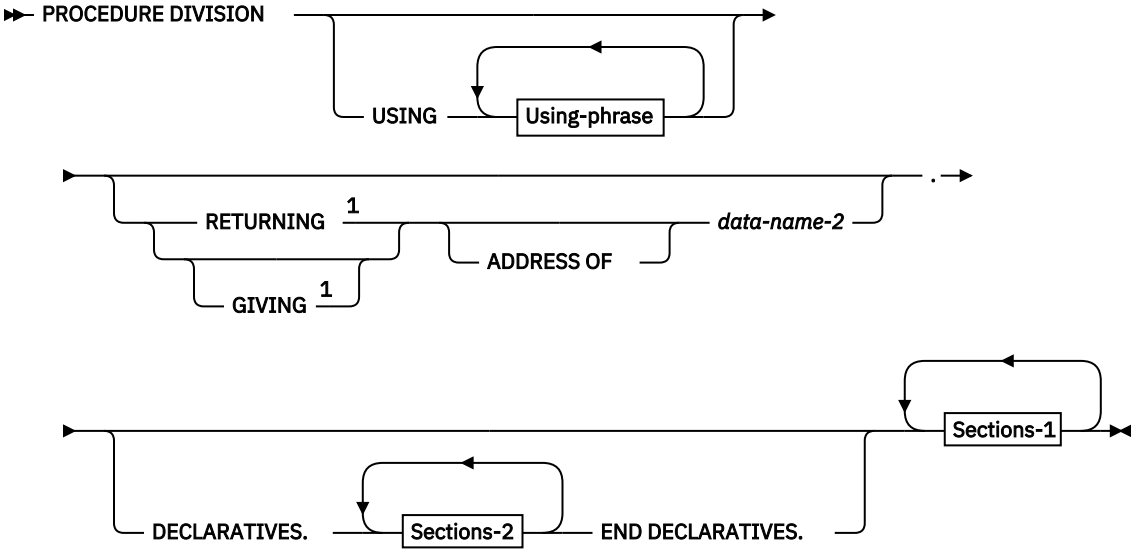
The structure of the Procedure Division is as follows:

- Format 1 - with Sections and Paragraphs
- Format 2 - with Paragraphs Only.

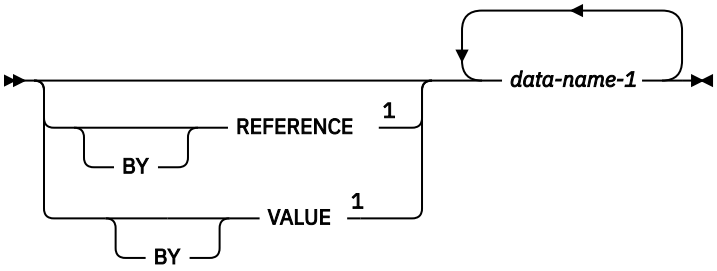
Execution begins with the first statement in the Procedure Division, excluding declaratives. Statements are executed in the order in which they are presented for compilation, unless the statement rules dictate some other order of execution.

The Procedure Division ends at the END PROGRAM header, before the beginning of the next COBOL source program, or at the physical end of the program. The physical end of the program is the physical position in a source program after which no further statements appear.

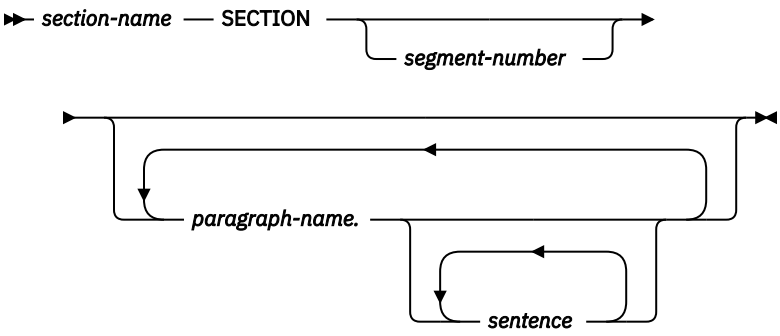
Format 1 - with Sections and Paragraphs



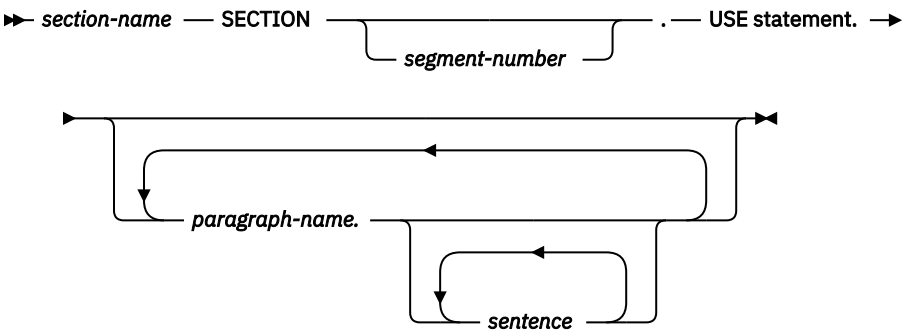
Using-phrase



Sections-1



Sections-2

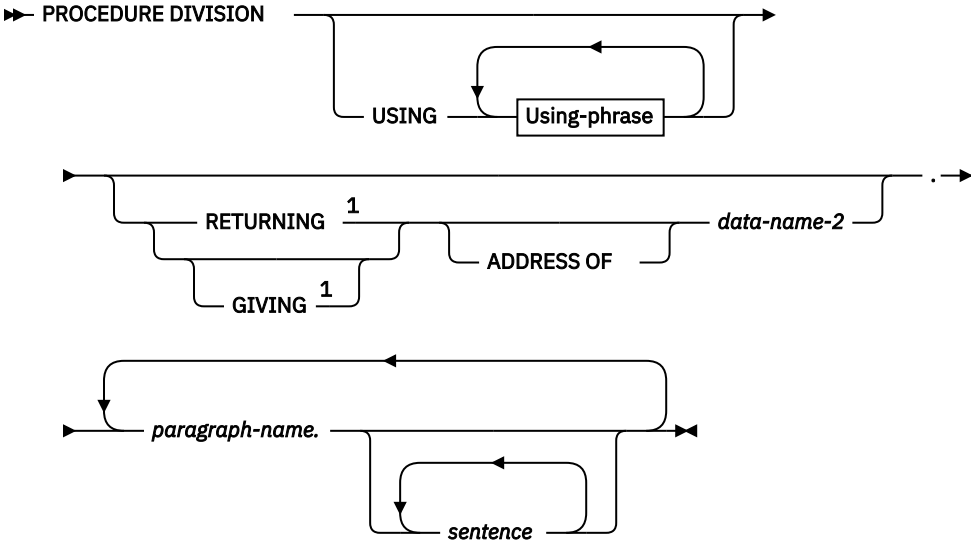


Procedure Division - Format 1

Notes:

[1] IBM Extension

Format 2 - with Paragraphs Only



Procedure Division - Format 2

Notes:

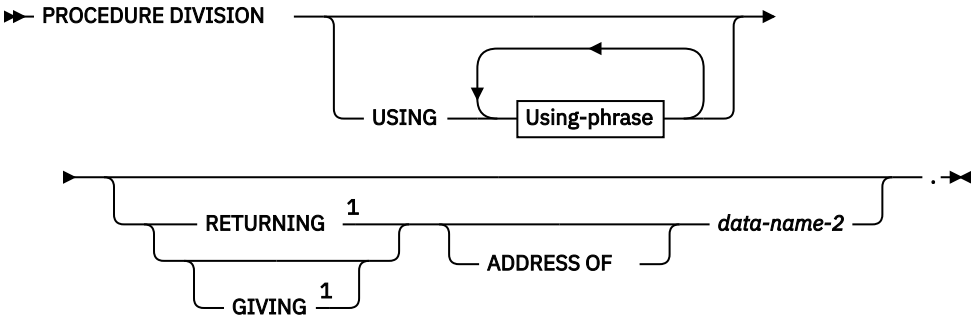
[1] IBM Extension

```
SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..

 004010 PROCEDURE DIVISION.
 004020 DECLARATIVES.
 004030 SECTION-NAME SECTION.
 004040 PARAGRAPH-NAMES.
 004050     PROGRAMMING STATEMENTS.
 004060*    COMMENTS.
 004070 END DECLARATIVES.
 004080 SECTION-NAME SECTION.
 004090 PARAGRAPH-NAME.
 004100     PROGRAMMING STATEMENTS.
```

Figure 14. Coding Example to Show Procedure Division Organization

The Procedure Division Header

The Procedure Division, if specified, is identified by the following header.



Procedure Division - Header - Format

Notes:

[1] IBM Extension

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 USING phrase makes data items defined in a calling program available to a called subprogram.

The following rules for the USING phrase assume that the calling and called programs are written in COBOL.

- 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. 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.
- Each USING identifier must be defined as a level-01 or level-77 item in the Linkage Section of the called subprogram
- A USING identifier must not contain a REDEFINES clause
- A particular user-defined word cannot 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 255 when a program is called with a LINKAGE TYPE of program. For programs called with LINKAGE TYPE of procedure, the maximum number of data-names is 400.
- 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 can be referred to in the called program; a given identifier can appear more than once. These items are defined in any Data Division section.
- A USING identifier containing the GLOBAL clause can be specified in only one Procedure Division header in a compilation unit.
- [IBM Extension] An identifier can 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.

[IBM Extension] BY REFERENCE

The BY REFERENCE phrase applies to all parameters that follow until overridden by another BY REFERENCE or BY VALUE phrase.

When a CALL argument is passed BY CONTENT or by REFERENCE, BY REFERENCE must be specified or implied for the corresponding formal parameter on the PROCEDURE DIVISION header.

BY REFERENCE is the default if neither BY REFERENCE or BY VALUE is specified.

You can use the BY REFERENCE phrase to pass an internal or external floating-point, DBCS, date, time, or timestamp data item.

[End of IBM Extension]

[IBM Extension] BY VALUE

The BY VALUE phrase applies to all parameters that follow until overridden by another BY VALUE or BY REFERENCE phrase.

You can use the BY VALUE phrase to pass an internal or external floating-point, date, time, or timestamp data item.

[End of IBM Extension]

[IBM Extension] GIVING/RETURNING Phrase

GIVING and RETURNING are equivalent.

[End of IBM Extension]

data-name-2

Data-name-2 is an output-only parameter. It specifies a data item to be returned as a program result. You must define data-name-2 in the LINKAGE or WORKING-STORAGE section. It can not be subscripted or reference modified.

Data-name-2 can be an internal or external floating-point, DBCS, date, time, or timestamp data item.

When a program returns to its invoker, the value in data-name-2 is implicitly stored into the identifier specified in the CALL RETURNING phrase.

The existence of the RETURNING phrase has no effect on the setting of the RETURN-CODE special register.

If the calling program is COBOL, it must specify the GIVING/RETURNING phrase of the CALL statement. In addition, data-name-2 and the corresponding CALL RETURNING identifier in the calling program must have the same number of character positions and must be of the same USAGE clause, SIGN clause and category.

Do not use the PROCEDURE DIVISION RETURNING phrase in main programs. The results are unpredictable. You should only specify the PROCEDURE DIVISION RETURNING phrase on called subprograms. For main programs, use the RETURN-CODE special register to return a value to the operating environment.

Items referenced in the RETURNING/GIVING phrase of the PROCEDURE DIVISION header cannot contain the TYPE phrase.

ADDRESS OF special register

For information about this special register, see page “ADDRESS OF Special Register” on page 126.

Declaratives

Declaratives provide one or more special-purpose sections that are executed when an exception-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 exception condition occurs. Each Declarative Section ends with another section-name followed by a USE sentence, or with the keywords END DECLARATIVES. See “USE Statement” on page 533 for more information on the USE statement. See “Precedence Rules for Nested Programs” on page 535 on using the GLOBAL phrase.

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 keywords END DECLARATIVES. The keywords 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, a USE sentence, and 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 caused the execution of it.

Within a declarative procedure, there must be no reference to any nondeclarative procedure.

A procedure-name associated with a USE statement can be referenced in a different declarative section or in a nondeclarative procedure only with a PERFORM statement.

A declarative is run as a separate invocation from any other declaratives or from the nondeclarative part of the COBOL program. See the section on using declaratives in the error handling chapter of the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

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.

Note: A COBOL procedure should not be confused with an ILE procedure (an ILE COBOL source program).

A **procedure-name** is a user-defined name that identifies a section or a paragraph.

Section

A **section** consists of a section header optionally followed by one or more paragraphs. A **section-header** is a section-name followed by: the keyword SECTION, an optional segment-number, and a separator period. The section-header must begin in Area A. Segment-numbers are explained in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

A **section-name** is 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 keywords END DECLARATIVES.

Paragraph

A **paragraph** consists of a paragraph-name followed by a separator period, optionally followed by one or more sentences.

A **paragraph-name** is 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. In the declaratives portion, a paragraph ends immediately before the next paragraph, the next USE statement, or at the keywords END DECLARATIVES. If one paragraph in a program is contained within a section, all paragraphs of the program must be contained in sections.

Sentence

A **sentence** consists of one or more statements terminated by a separator period.

Statement

A **statement** is a syntactically valid combination of identifiers and symbols (literals, relational-operators, and so forth) beginning with a COBOL verb.

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.

Identifier

An **identifier** is 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 or function argument in a test intrinsic function), the contents of an identifier must be compatible with the class specified through its PICTURE or FORMAT clause, or results are unpredictable.

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

Expressions calculate values which can then be used as operands in conditional and arithmetic statements. Arithmetic expressions are built up from operands and operators under a strict hierarchy and precedence.

In general, any arithmetic expression can be:

1. An elementary numeric item such as:
   - A numeric literal (integer or decimal)
   - An identifier describing an elementary numeric item
   - The figurative constant ZERO (ZEROS, ZEROES)
   - Numeric functions
2. An arithmetic expression surrounded by parentheses
3. An arithmetic expression preceded by a unary operator (+, -)
4. Two arithmetic expressions separated by a binary arithmetic operator (+, -, *, /, **)

Identifiers and literals appearing in arithmetic expressions must represent either numeric elementary items or numeric literals on which arithmetic may be performed.

Exponential Expressions

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, always results in +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.

Unless the exponent is a literal integer with a value of 2, 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.

Arithmetic Operators

Five binary and two unary arithmetic operators can be used in arithmetic expressions. They are represented by specific characters that must be preceded and followed by a space.

Binary Operator
Meaning

+
Addition

-
Subtraction

*****
Multiplication

/
Division

******
Exponentiation

Unary Operator
Meaning

+
Multiplication by +1

-
Multiplication by -1

Parentheses are used to highlight or modify the order of evaluation of complex expressions. This improves both readability and maintainability.

Left and right parentheses must be paired in an arithmetic expression with the left parenthesis appearing before its corresponding right parenthesis.

Expressions within parentheses are evaluated first and parenthetical pairs can be nested within other pairs. Evaluation proceeds from the least inclusive pairing outward.

When the order of evaluation is not made explicit by parentheses, expressions are evaluated left-to-right following the hierarchy listed below:

1. Unary operator
2. Exponentiation
3. Multiplication and division
4. Addition and subtraction.

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.

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 21 on page 225 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 21. Valid Arithmetic Symbol Pairs					
First Symbol	**Second 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

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
- 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

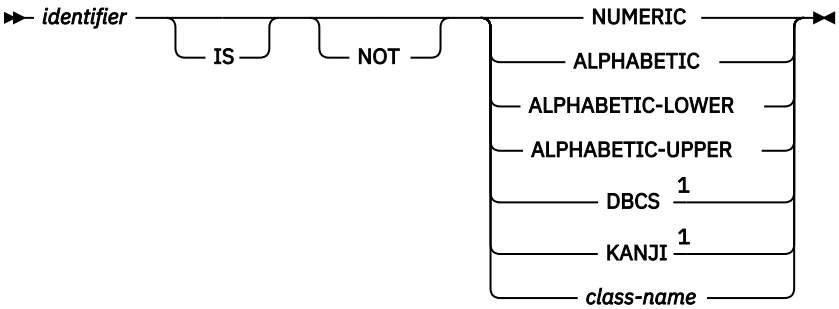
There are five simple conditions:

- 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.



Class Condition - Format
Notes:

[1] IBM Extension

identifier
Must reference a data item whose usage is DISPLAY. When the identifier is a group item of zero length and NOT is specified in the class condition, the result is always true. If NOT is not specified, the result is always false.

If identifier is a function identifier, it must reference an alphanumeric, DBCS, or date-time function.

NOT
When used, NOT and the next keyword define the class test to be executed for truth value. For example, NOT NUMERIC is a truth test for determining that a data item 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 character set, 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).

[IBM Extension] For numeric and date-time 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 referenced by the identifier consists entirely of any combination of the lowercase or uppercase alphabetic characters A through Z, and the space.

ALPHABETIC-LOWER
The data item referenced by the identifier consists entirely of any combination of the lowercase alphabetic characters a through z, and the space.

ALPHABETIC-UPPER
The data item referenced by the identifier consists entirely of any combination of the uppercase alphabetic characters A through Z, and the space.

class-name
The data item referenced by the identifier 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.

[IBM Extension]

DBCS
Identifier consists entirely of DBCS characters, with the following rules:

- For DBCS data items, the identifier being tested must be described explicitly or implicitly as USAGE DISPLAY-1.
- A range check is performed on the data portion of the item for valid DBCS character representation. The valid range is X'41' through X'FE' for both bytes. X'4040' is a DBCS blank.

KANJI
Identifier consists entirely of DBCS characters, with the following rules:

- For DBCS data items, the identifier being tested must be described explicitly or implicitly as USAGE DISPLAY-1.

- A range check is performed on the data portion of the item for valid DBCS character representation. The valid range is X'41' through X'7F' for the first byte, and X'41' through X'FE' for the second byte. X'4040' is a DBCS blank.

[End of IBM Extension]

The class test is not valid for items whose usage is INDEX, POINTER, or PROCEDURE-POINTER because these items do not belong to any class or category.

[IBM Extension] The class condition cannot be used for external floating-point (USAGE DISPLAY) or internal floating-point (USAGE COMP-1 and USAGE COMP-2) items. [End of IBM Extension]

Table 22 on page 227 shows valid forms of the class test.

Table 22. 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 Internal-Decimal	NUMERIC	NOT NUMERIC
[IBM Extension] DBCS DBCS-edited [End of IBM Extension]	DBCS KANJI	NOT DBCS NOT KANJI
[IBM Extension] Date-Time [End of IBM Extension]	NUMERIC class-name	NOT NUMERIC NOT class-name

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.

Condition-Name Condition - 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.

[IBM Extension] Condition-names with floating-point and DBCS values are allowed. [End of IBM Extension]

The following example illustrates the use of conditional variables and condition-names:

```
01  NUMBER              PIC  99.
    88  FIVE            VALUE 5.
    88  ONE-DIGIT-EVEN   VALUE 0, 2, 4, 6, 8
    88  TWO-DIGIT-NUMBER VALUE 10 THRU 99
```

NUMBER is the conditional variable; FIVE, ONE-DIGIT-EVEN, TWO-DIGIT-NUMBER are condition-names.

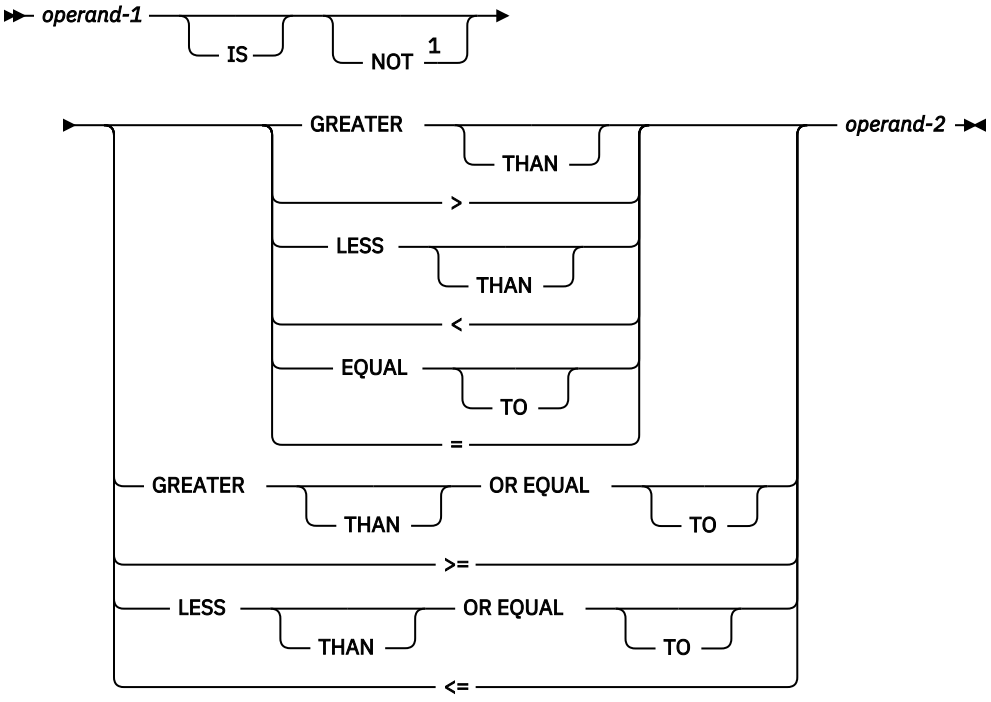
The following IF statements can be added to the above example to determine the age group of a specific record:

```
IF FIVE...             (Tests for value 5)
IF ONE-DIGIT-EVEN      (Tests for values 0, 2, 4, 6, 8)
IF TWO-DIGIT-NUMBER    (Tests for values 10 thru 99)
```

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, index-name or a function-identifier. The relation condition must contain at least one reference to an identifier.



Relation Condition - Format

Notes:

[1] NOT GREATER THAN OR EQUAL TO, NOT >=, NOT LESS THAN OR EQUAL TO, and NOT <=, are IBM Extensions.

operand-1

The subject of the relation condition. Can be an identifier, literal, function-identifier, arithmetic expression, or index-name.

operand-2
The object of the relation condition. Can be an identifier, literal, function-identifier, arithmetic expression, or index-name.

The relational operator specifies the type of comparison to be made. Each relational operator must be preceded and followed by a space.

Relational Operator	Can Be Written
IS GREATER THAN	IS **>**
IS NOT GREATER THAN	IS NOT **>**
IS LESS THAN	IS **<**
IS NOT LESS THAN	IS NOT **<**
IS EQUAL TO	IS **=**
IS NOT EQUAL TO	IS NOT **=**
IS GREATER THAN OR EQUAL TO	IS **>=**
IS LESS THAN OR EQUAL TO	IS **<=**

[IBM Extension]

IS NOT GREATER THAN OR EQUAL TO	IS NOT **>=**
IS NOT LESS THAN OR EQUAL TO	IS NOT **<=**

[End of IBM Extension]

[IBM Extension] DBCS Items

DBCS data items and literals can be used with all relational operators. Comparisons (between two DBCS items only) are based on the binary collating sequence of the hexadecimal values of the DBCS characters. If the items are not of the same length, the smaller item is padded with DBCS spaces to the right.

[End of IBM Extension]

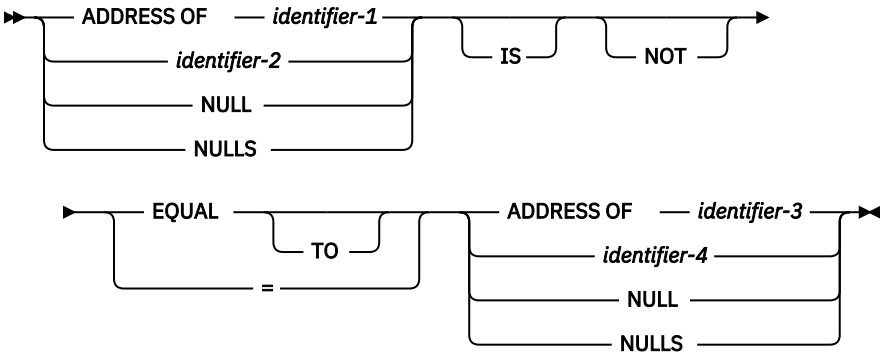
[IBM Extension] Pointer Data Items

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.

Only EQUAL and NOT EQUAL are allowed as relational operators when you specify pointer data items. 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.

ADDRESS Comparison - Format



identifier-1, identifier-3
May specify any level item defined in the Data Division Section, except level 66 and level 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]

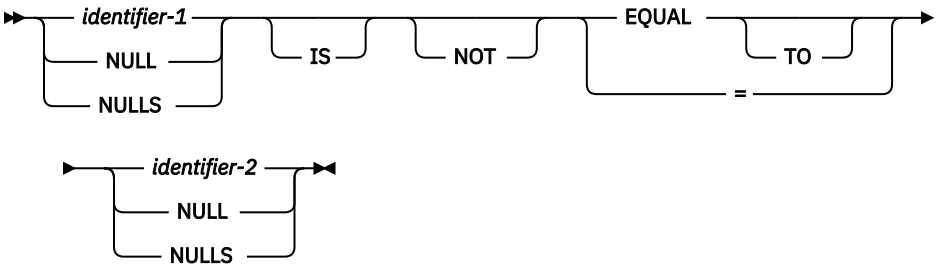
[IBM Extension] Procedure-pointer Data Items

Procedure-pointer data items are items defined explicitly as USAGE IS PROCEDURE-POINTER.

Only EQUAL and NOT EQUAL are allowed as relational operators when you specify procedure-pointer data items. 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 procedure-pointer data items.

Procedure-Pointer Comparison - Format



identifier-1, identifier-2
Must be described as USAGE IS PROCEDURE-POINTER (see “PROCEDURE-POINTER Phrase” on page 211 for more information).

NULL(S)
Used only if the other operand is defined as USAGE IS PROCEDURE-POINTER. NULL=NULL is not allowed.

[End of IBM Extension]

Comparison of Numeric and Nonnumeric Operands

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 23 on page 231 summarizes permissible comparisons with *nonnumeric* operands.

Table 24 on page 232 summarizes permissible comparisons with *numeric* operands.

The symbols used in Table 23 on page 231 and Table 24 on page 232 are as follows:

- NN = Comparison for nonnumeric operands.
- NU = Comparison for numeric operands.
- NL = Comparison for national operands.
- NLN = Comparison for national and nonnational operands.
- DT = Comparison for date-time operands.
- Blank = Comparison is not allowed.

Table 23. Permissible Comparisons with Nonnumeric Second Operands

FIRST OPERAND	**SECOND OPERAND**												
	GR	**AL**	**AN**	**ANE**	**NE**	**FC[1]**	**NNL**	**DB**	**DBE**	**DA**	**TI**	**TS**	**NL**
NONNUMERIC OPERAND													
Group (GR)	NN	NN	NN	NN	NN	NN	NN			NN	NN	NN	NN
Alphabetic (AL)	NN	NN	NN	NN	NN	NN	NN						NLN
Alphanumeric (AN)	NN	NN	NN	NN	NN	NN	NN			NN	NN	NN	NLN
Alphanumeric Edited (ANE)	NN	NN	NN	NN	NN	NN	NN			NN	NN	NN	
Numeric Edited (NE)	NN	NN	NN	NN	NN	NN	NN			NU	NU	NU	
Figurative Constant (FC[1])	NN	NN	NN	NN	NN								NL[4]
Nonnumeric Literal (NNL)	NN	NN	NN	NN	NN					NN	NN	NN	NLN
DBCS items (DB)[3]								NN	NN				NLN
DBCS-edited items (DBE)[3]								NN	NN				
Date (DA)[3]	NN		NN	NN	NU		NN			DT		DT	
Time (TI)[3]	NN		NN	NN	NU		NN				DT	DT	
Timestamp (TS)[3]	NN		NN	NN	NU		NN			DT	DT	DT	
National (NL)	NN	NLN	NLN			NL[4]	NLN	NLN					NL
NUMERIC OPERAND													
Figurative Constant ZERO (ZR)	NN	NN	NN	NN	NN								
Numeric Literal (NL)	NN	NN	NN	NN	NN					NU	NU	NU	
External Decimal (ED)[2]	NN	NN	NN	NN	NN	NN	NN			NU	NU	NU	NN
Binary (BI)										NU	NU	NU	
Arithmetic Expression (AE)										NU	NU	NU	

Table 23. Permissible Comparisons with Nonnumeric Second Operands (continued)

FIRST OPERAND	SECOND OPERAND												
	GR	**AL**	**AN**	**ANE**	**NE**	**FC[1]**	**NNL**	**DB**	**DBE**	**DA**	**TI**	**TS**	**NL**
Boolean data item or Boolean Literal (BO)[3]													
Internal Decimal (ID)										NU	NU	NU	
Internal Floating-Point (IFP)[3]													
External Floating-Point (EFP)[3]	NN[3]	NN[3]	NN[3]	NN[3]	NN[3]	NN[3]	NN[3]						
Floating-Point Literal (FPL)[3]													
National Decimal (ND) [23]	NN	NN	NN	NN	NN	NN	NN			NU	NU	NU	NN

Table 24. Permissible Comparisons with Numeric Second Operands

FIRST OPERAND	SECOND OPERAND									
ZR	NL	ED, ND	BI	AE	BO	ID	IFP[3]	EFP[3]	FPL[3]	
NONNUMERIC OPERAND										
Group (GR)	NN	NN	NN[2]						NN[3]	
Alphabetic (AL)	NN	NN	NN[2]						NN[3]	
Alphanumeric (AN)	NN	NN	NN[2]						NN[3]	
Alphanumeric Edited (ANE)	NN	NN	NN[2]						NN[3]	
Numeric Edited (NE)	NN	NN	NN[2]						NN[3]	
Figurative Constant (FC[1])			NN[2]						NN[3]	
Nonnumeric Literal (NNL)			NN[2]						NN[3]	
Date (DA)		NU	NU[2]	NU	NU		NU			
Time (TI)		NU	NU[2]	NU	NU		NU			
Timestamp (TS)		NU	NU[2]	NU	NU		NU			
NUMERIC OPERAND										
Figurative Constant ZERO (ZR)			NU	NU	NU	NU[3]	NU	NU[3]	NU[3]	
Numeric Literal (NL)			NU	NU	NU		NU	NU[3]	NU[3]	
External Decimal (ED)	NU	NU	NU	NU	NU		NU	NU[3]	NU[3]	NU[3]
Binary (BI)	NU	NU	NU	NU	NU		NU	NU[3]	NU[3]	NU[3]
Arithmetic Expression (AE)	NU	NU	NU	NU	NU		NU	NU[3]	NU[3]	NU[3]
Boolean data item or Boolean Literal (BO)[3]	NU[3]					NU[3]				
Internal Decimal (ID)	NU	NU	NU	NU	NU		NU	NU[3]	NU[3]	NU[3]
Internal Floating-Point (IFP)[3]	NU[3]	NU[3]	NU[3]	NU[3]	NU[3]		NU[3]	NU[3]	NU[3]	NU[3]

Table 24. Permissible Comparisons with Numeric Second Operands (continued)

FIRST OPERAND	SECOND OPERAND									
External Floating-Point (EFP)[3]	NU[3]	NU[3]	NU[3]	NU[3]	NU[3]		NU[3]	NU[3]	NU[3]	NU[3]
Floating-Point Literal (FPL)[3]			NU[3]	NU[3]	NU[3]		NU[3]	NU[3]	NU[3]	
National Decimal (ND) [23]	NU	NU	NU	NU	NU		NU	NU[3]	NU[3]	NU

Notes to Table 23 on page 231 and Table 24 on page 232:

1 Includes all figurative constants except ZERO and NULL

2 Integer item only

3 IBM extension

4 Only for SPACE

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. 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 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.

- [IBM Extension] External floating-point items can be compared with nonnumeric operands. [End of IBM Extension]

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 338.)

[IBM Extension] Comparing Boolean Operands

Boolean operands are 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.

[End of IBM Extension]

[IBM Extension] Comparing DBCS Operands

The rules for comparing DBCS or DBCS-edited operands are the same as those for the comparison of nonnumeric operands. The comparison is based on a binary collating sequence of the hexadecimal values of the DBCS characters. The PROGRAM COLLATING clause of the OBJECT-COMPUTER paragraph will have no effect on this.

[End of IBM Extension]

[IBM Extension] Comparing National Operands

An operand of class NATIONAL may be compared to another operand of class NATIONAL. The result of such a comparison is based on the binary collating sequence of the hexadecimal values of the UCS-2 character set. If two operands have unequal size, the shorter one is padded to the right with the padding character specified in the Padding Character compile option or the equvilant process option. The default is the UCS-2 double-byte space character (NX"3000").

[End of IBM Extension]

[IBM Extension] Comparing National and Non-National Operands

An operand of class NATIONAL may be compared with an Alphabetic data item, an Alphanumeric data item, a DBCS data item, a Nonnumeric literal, or a DBCS literal.

The data item or literal that is not a national item is treated as though it were moved, in accordance with the rules of the MOVE statement, to an elementary data item of class national and the same logical length. The converted value is then compared to the national operand. If the items are of different length, the shorter item is padded to the right with the padding character specified in the Padding Character compile option, or the equivalant process option. The default is the UCS-2 single-byte space character if the non-national operand is a single-byte item, or the UCS-2 double-byte space character (NX"3000") if the non-national operand is a double-byte item.

[End of IBM Extension]

[IBM Extension] Comparing Date-Time Operands

If an item of class date-time is compared to a nonnumeric operand (except for numeric-edited operands), the date-time item is treated as if it were nonnumeric.

During the comparison of an item of class date-time to an numeric-edited or numeric operand, the date-time item is de-edited. De-editing results in a numeric integer. This numeric integer is then numerically compared with the other operand.

During the comparison of one date-time item with another, the items are first converted to a common date, time, or timestamp format, and then compared. Characters that are part of a format literal, but which are not conversion specifiers (for example the / or - characters), have no effect on a date-time comparison.

When comparing a date item to a timestamp item, only the date portion of the timestamp is considered. When comparing a time item to a timestamp item, only the time portion of the timestamp is considered.

[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 a 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 of the index-name is compared with the data item or literal.
- [IBM Extension] In the comparison of an index-name with an arithmetic expression, the occurrence number that corresponds to the value of the index-name is compared with the arithmetic expression.

  Since an integer function may be used wherever an arithmetic expression may be used, this extension allows you to compare an index-name to an integer or numeric function.

  [End of IBM Extension]
- 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 25 on page 235 shows valid comparisons involving index-names and index data items.

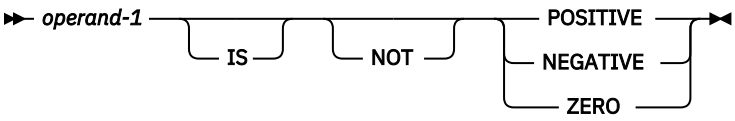
Table 25. Comparisons Involving Index Names and Index Data Items

Operands Compared	**Index-Name**	**Index Data Item**	**Data-Name**	**Literal**	**Arithmetic Expression**
Index-Name	Compare occurrence number	Compare without conversion	Compare occurrence number with data-name	Compare occurrence number with literal	Compare occurrence number with arithmetic expression
Index Data Item	Compare without conversion	Compare without conversion	Not permitted	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.

Sign Condition - 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.

[IBM Extension] The operand can be defined as a floating-point identifier. [End of IBM Extension]

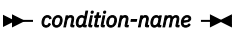
An unsigned operand is either POSITIVE or ZERO.

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, by testing 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.

Switch-Status Condition - 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 78.)

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 26. 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

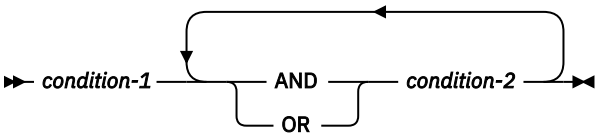
A simple condition is negated through the use of the logical operator NOT. The negated simple condition gives the opposite truth value of the simple condition.

Negated Simple Condition - Format

Combined Conditions

Two or more conditions can be logically connected to form a combined condition.

Combined Conditions - 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 the following table.

Table 27. Combined Conditions—Permissible Element Sequences

Combined condition element	Leftmost	When not leftmost, can be immediately preceded by:	Rightmost	When not rightmost, can 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 (

Table 27. Combined Conditions—Permissible Element Sequences (continued)				
Combined condition element	**Leftmost**	**When not leftmost, can be immediately preceded by:**	**Rightmost**	**When not rightmost, can be immediately followed by:**
)	No	simple-condition)	Yes	OR AND)

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.

The following table illustrates the relationships between logical operators and conditions C1 and C2.

Table 28. Logical Operators and Evaluation Results of Combined Conditions							
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

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.

Values are established for arithmetic expressions and functions if and when the conditions containing them are evaluated. Similarly, negated conditions are evaluated if and when it is necessary to evaluate the complex condition that they represent.

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))
```

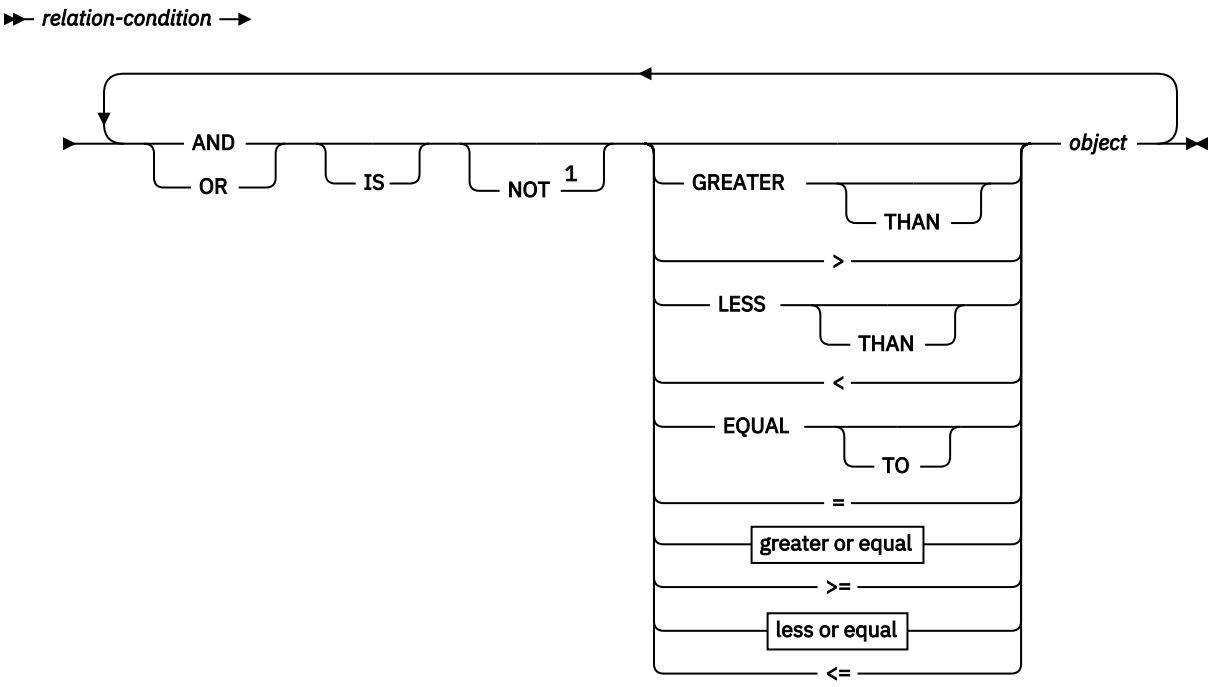
The order of evaluation in this example is as follows:

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

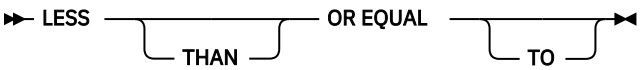
When relation-conditions are written consecutively without intervening parentheses, any relation-condition after the first can be abbreviated in one of two ways:

- Omission of the subject
- Omission of the subject and relational operator.



greater or equal

less or equal



Abbreviated Combined Relation Conditions - Format

Notes:

[1] NOT GREATER THAN OR EQUAL TO, NOT >=, NOT LESS THAN OR EQUAL TO, and NOT <=, are IBM Extensions.

An object is any data item or expression that can be compared to the subject of the preceding relation condition.

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 Table 29 on page 241.

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 29. 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 243). A series of imperative statements can be specified whenever an imperative statement is allowed.

Table 30 on page 241 lists COBOL imperative statements.

Table 30. Types of Imperative Statements	
Type	**Imperative Statement**
Arithmetic	ADD[1] COMPUTE[1] DIVIDE[1] INSPECT (TALLYING) MULTIPLY[1] SUBTRACT[1]
Data Manipulation	ACCEPT (DATE, DAY, DAY-OF-WEEK, TIME) INITIALIZE INSPECT (CONVERTING) INSPECT (REPLACING) MOVE SET STRING[2] UNSTRING[2] [IBM Extension] XML GENERATE[6] XML PARSE[6] [End of IBM Extension]

Table 30. Types of Imperative Statements (continued)	
Type	**Imperative Statement**
Ending	STOP RUN EXIT PROGRAM [IBM Extension] GOBACK [End of IBM Extension]
Input/Output	ACCEPT[6] identifier CLOSE DELETE[3] DISPLAY[6] OPEN READ[4] REWRITE[3] SET (for UPSI switches) 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 30 on page 241:

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 225.)

Figure 15 on page 243 lists COBOL statements that are conditional, or that become conditional when a condition is included (for example: ON SIZE ERROR, or ON OVERFLOW) and the statement is not terminated by its explicit scope terminator.

```
Arithmetic                        Ordering
ADD...ON SIZE ERROR               RETURN...AT END
ADD...NOT ON SIZE ERROR           RETURN...NOT AT END
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
Data Manipulation                 Subprogram Linkage
STRING...ON OVERFLOW              CALL...ON OVERFLOW
STRING...NOT ON OVERFLOW          CALL...ON EXCEPTION
UNSTRING...ON OVERFLOW            CALL...NOT ON EXCEPTION
UNSTRING...NOT ON OVERFLOW
 [IBM Extension] XML GENERATE...ON EXCEPTION
XML GENERATE...NOT ON EXCEPTION
XML PARSE...ON EXCEPTION
XML PARSE...NOT ON EXCEPTION [End of IBM Extension]
Decision                          Table Handling
IF                                SEARCH...WHEN
EVALUATE
Input/Output
ACCEPT...ON EXCEPTION                    READ...NO DATA
ACCEPT...NOT ON EXCEPTION                REWRITE...INVALID KEY
DELETE...INVALID KEY                     REWRITE...NOT INVALID KEY
DELETE...NOT INVALID KEY                 START...INVALID KEY
DISPLAY...ON EXCEPTION                   START...NOT INVALID KEY
DISPLAY...NOT ON EXCEPTION               WRITE...AT END~OF~PAGE
READ...AT END                            WRITE...NOT AT END~OF~PAGE
READ...NOT AT END                        WRITE...INVALID KEY
READ...INVALID KEY                       WRITE...NOT INVALID KEY
READ...NOT INVALID KEY
```

Figure 15. Conditional Statements

Delimited Scope Statements

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

[IBM Extension] END-XML [End of IBM Extension]

Implicit Scope Terminators

The separator period at the end of any sentence is an implicit scope terminator that terminates the scope of any previous statement that is not yet terminated. When a statement is contained within another statement, the next phrase of the containing statement following the contained statement is an implicit scope terminator that ends the scope of the contained statement.

A conditional statement not terminated by its scope terminator cannot be contained within another statement.

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

These are statements that direct the compiler to take a specified action. They are discussed in Chapter 8, "Compiler-Directing Statements," on page 507.

Type	Compiler-Directing Statement
Library	COPY
Declarative	USE
Documentation	ENTER
Compiler options	PROCESS
Source text	REPLACE
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
- Ordering

- Subprogram linkage
- Table handling
- Procedure branching

Common Phrases and Concepts

There are several phrases and concepts common to arithmetic and data manipulation statements:

- CORRESPONDING phrase
- GIVING phrase
- ROUNDED phrase
- SIZE ERROR phrase
- Overlapping operands

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 keyword CORRESPONDING must be group items. In this discussion, these identifiers are referred to as identifier-1 and identifier-2.

A pair of data items (subordinate items), one from identifier-1 and one from identifier-2, 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 identifier-1 and identifier-2.
- The subordinate items are not identified by the keyword FILLER.
- Neither identifier-1 nor identifier-2 is described as a level 66 or 88 item, nor is the usage of either item INDEX, POINTER, or PROCEDURE-POINTER. Neither identifier-1 nor identifier-2 can be reference modified. The name of the data item must be unique after application of the implied qualifiers.
- The subordinate items do not include a REDEFINES, RENAMES, OCCURS, USAGE IS INDEX, USAGE IS POINTER, or USAGE IS PROCEDURE-POINTER clause in their descriptions; if such a subordinate item is a group, the items subordinate to it are also ignored.

  However, identifier-1 and identifier-2 themselves may contain or be subordinate to items containing a REDEFINES or OCCURS clause in their descriptions.
- Identifier-1 and identifier-2 can be subordinate to a FILLER item.

  For example, if two data hierarchies are defined as follows:

```
05  ITEM-1 OCCURS 6 INDEXED BY X.
  10  ITEM-A PIC S9(3).
  10  ITEM-B PIC 99V9.
  10  ITEM-C PIC X(4).
  10  ITEM-D REDEFINES ITEM-C PIC 9(4).
  10  ITEM-E PIC 9(4) USAGE COMP.
  10  ITEM-F USAGE INDEX.
  10  ITEM-G PIC X(4).
05  ITEM-2.
  10  ITEM-A PIC 99.
  10  ITEM-B PIC 9V9.
  10  ITEM-C PIC A(4).
  10  ITEM-D PIC 9(4).
  10  ITEM-E PIC 9(9) USAGE COMP.
  10  ITEM-F USAGE INDEX.
  10  ITEM-G PIC X(4).
```

Then, if ADD CORR ITEM-2 TO ITEM-1(X) is specified,

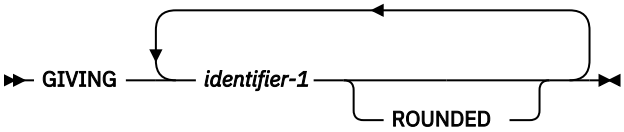
- ITEM-A and ITEM-A(X); ITEM-B and ITEM-B(X); ITEM-E and ITEM-E(X) are considered to be corresponding and are added together
- ITEM-C and ITEM-C(X); ITEM-G and ITEM-G(X) are not included, because they are not numeric
- ITEM-D and ITEM-D(X) are not included, because ITEM-D(X) includes a REDEFINES clause in its data description.
- ITEM-F and ITEM-F(X) are not included, because they are defined as USAGE IS INDEX

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.

GIVING Phrase

The data item referenced by the identifier that follows the word GIVING is set to the calculated result of the arithmetic operation. Because this identifier is not involved in the computation, it may be a numeric edited item.

Giving Phrase - 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.

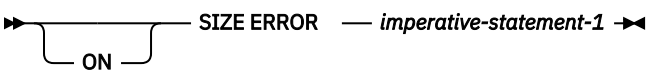
When the size of the fractional result exceeds the number of places provided for its storage, truncation occurs unless ROUNDED is specified. When ROUNDED is specified, the least significant digit of the resultant identifier is increased by 1 whenever the most significant digit of the excess is greater than or equal to 5. The maximum number of digits that can be accurately rounded is 62.

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.

[IBM Extension] In a floating-point arithmetic operation, the ROUNDED phrase has no effect; the result of a floating-point operation is *always* rounded. For more information on floating-point arithmetic expressions, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*. [End of IBM Extension]

SIZE ERROR Phrases

SIZE ERROR Phrase - Format



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
- In an exponential expression, when
  - Zero is raised to the exponent zero

- Zero is raised to a negative exponent
- A negative number is raised to a fractional exponent

The size error condition applies only to final results, not to any intermediate results.

If the resultant identifier is defined with USAGE IS BINARY, COMP-4, or COMP-5, the largest value that can be contained in it is the maximum value implied by its associated decimal PICTURE character-string.

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** 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. Values of other resultant identifiers are not affected, as long as no size error occurred for them. After completion of the execution of the arithmetic operation, the imperative statement in the ON SIZE ERROR phrase is executed. If no explicit transfer of control is executed upon completion of the imperative statement specified in the ON SIZE ERROR phrase, control is transferred to the end of the arithmetic statement and the NOT ON SIZE ERROR phrase, if specified, is ignored.

If the ON SIZE ERROR phrase is not specified and a size error condition exists after the execution of the arithmetic operations specified by an arithmetic statement, the value of the affected resultant identifier is undefined. Values of other resultant identifiers are not affected, as long as no size error occurred for them. After completion of the arithmetic operations, control is transferred to the end of the arithmetic statement and the NOT ON SIZE ERROR phrase, if specified, is ignored.

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.

NOT ON SIZE ERROR

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.

Overlapping Operands

When a sending and a receiving item in any statement share a part or all of their storage areas, yet are not defined by the same data description entry, the result of the execution of such a statement is unpredictable. In addition, the results are unpredictable for some statements in which sending and receiving items are defined by the same data description entry. These cases are addressed in the general rules associated with those statements.

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.

[IBM Extension] The maximum size of a zoned decimal or an internal decimal operand is 63 decimal digits. [End of IBM Extension]

From each operand, you can determine the number of decimal digits in the **composite of operands**. 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.

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).
```

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]

When the (default) compiler option *NOEXTEND or the PROCESS statement option NOEXTEND is specified, the composite of operands can have a maximum length of 30 decimal digits.

The composite of operands can have a maximum length of 31 decimal digits when the arithmetic mode compiler option *EXTEND31 or PROCESS statement option EXTEND31 is specified.

The composite of operands can have a maximum length of 34 decimal digits when the arithmetic mode compiler option *EXTEND31FULL or PROCESS statement option EXTEND31FULL is specified.

The composite of operands can have a maximum length of 63 decimal digits when the arithmetic mode compiler option *EXTEND63 or PROCESS statement option EXTEND63 is specified.

Note: If the composite of operands exceeds the specified maximum, significant digits may be lost during execution.

[End of IBM Extension]

The following table shows the maximum number of decimal digits that are allowed for the composite of operands in arithmetic statements.

Compiler Option/Process Statement	Maximum Length of Composite (decimal digits)
*NOEXTEND/NOEXTEND	18 [IBM Extension] 30 [End of IBM Extension]
*EXTEND31/EXTEND31	18 [IBM Extension] 31 [End of IBM Extension]
*EXTEND31FULL/EXTEND31FULL	[IBM Extension] 34 [End of IBM Extension]
*EXTEND63/EXTEND63	[IBM Extension] 63 [End of IBM Extension]

The following list shows how the composite of operands is determined for arithmetic statements:

Statement
Determination of the Composite of Operands

ADD
Superimposing all operands in a given statement (except those following the word GIVING)

COMPUTE
Restriction does not apply

DIVIDE
Superimposing all receiving data items, except the REMAINDER data item

MULTIPLY
Superimposing all receiving data items

SUBTRACT
Superimposing all operands in a given statement (except those following the word GIVING)

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. For more information on arithmetic precision, see "Appendix B. Intermediate Results and Arithmetic Precision" on page 539.

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.

Note: Intermediate results generated during the execution of arithmetic statements are system-specific and can affect program portability. Use of the individual arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE, rather than COMPUTE, reduces the risk of getting inconsistent results.

Data Manipulation Statements

The following COBOL statements move and inspect data: ACCEPT, INITIALIZE, INSPECT, MOVE, READ, RELEASE, RETURN, REWRITE, SET, STRING, UNSTRING, and WRITE.

[IBM Extension] XML PARSE, XML GENERATE. [End of IBM Extension]

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.

See "Appendix F. File Structure Support Summary and Status Key Values" on page 563 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 51 on page 568.

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 “Procedure Division Statements” on page 252.) 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, the following actions are taken:

1. 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.
2. Control will be transferred to the imperative statement of an INVALID KEY phrase, if specified.
3. 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.
4. 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.

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.

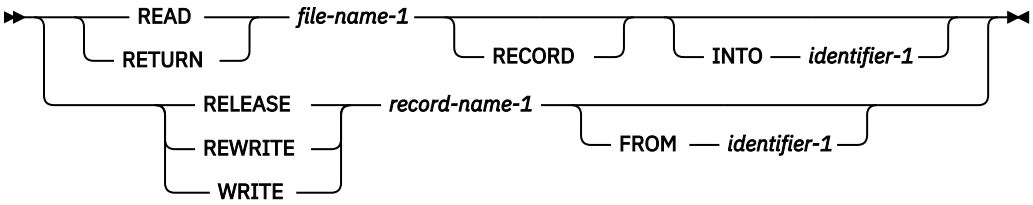
For more information about error handling and the role of the INVALID KEY phrase, see the chapter on exception and error handling in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

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, Local-Storage or the Linkage Sections, or

of a record description for another previously opened file. Record-name, identifier must not refer to the same storage area.

INTO/FROM Identifier Phrase - Format



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:

1. The execution of the same READ or RETURN statement without the INTO phrase.
2. The current record is moved from the record area to 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. If the file description entry contains a RECORD IS VARYING clause, the implied move is a group move. The implied MOVE statement does not occur if the execution of the READ or RETURN statement was unsuccessful. Any subscripting 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 result of the execution of a RELEASE, REWRITE, or WRITE statement with the FROM phrase is equivalent to the execution of the following statements in the order specified:

1. The statement

   ```
   MOVE identifier-1 TO record-name-1
   ```

   according to the rules specified for the MOVE statement.
2. The same RELEASE, REWRITE, or WRITE statement without the FROM phrase.

After the execution of the RELEASE, REWRITE or WRITE statement is complete, the information in 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 no longer exists, 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) and GLOBAL in the outermost program.

You may specify the DB-FORMAT-NAME special register in a function whenever an alphanumeric argument is allowed.

[End of IBM Extension]

Procedure Branching Statements

Statements, sentences, and paragraphs in the Procedure Division are executed sequentially, except when a procedure-branching statement (listed below) is used.

- ALTER
- EXIT
- GO TO
- PERFORM

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.

[IBM Extension]

- Format 3 - Feedback
- Format 4 - Local Data Area
- Format 5 - PIP Data Area
- Format 6 - Attribute Data Area
- Format 7 - Workstation I/O
- Format 8 - Session I/O
- Format 9 - Data Area

[End of IBM Extension]

Format 1 - Data Transfer

ACCEPT Statement - Format 1 - Data Transfer
Notes:

[1] IBM Extension

Format 1 transfers data from an input device into identifier-1. Incoming data is transferred as a character string aligned on the leftmost character position. No data conversion will occur. If the size of identifier-1 is greater than the record length of the input device, then additional data will be requested after the transfer of one record has been completed. The additional data will be transferred into identifier-1 starting at the position immediately to the right of the last character previously transferred from the device. This process will continue until identifier-1 has been filled. If on any transfer the device record holds more characters than are needed to fill identifer-1, then the excess data will be truncated.

Since all data is transferred as a character string, identifier-1 will normally be defined, explicitly or implicitly, with usage DISPLAY. The ACCEPT statement will, however, handle data in other formats, provided it is possible to enter the data on the input device in a format that corresponds to the internal representation of identifier-1.

Format 1 is useful for exception 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-1

The receiving data item.

[IBM Extension] If the description of identifier-1 contains a TYPE clause, the type-name referenced in that clause must be elementary. [End of IBM Extension]

[IBM Extension] Identifier-1 may be defined with usage DISPLAY-1, that is, it may be a DBCS or DBCS-edited item. The data on the input device must then be delimited by a shift-out and a shift-in character; these will be removed when the data is transferred. [End of IBM Extension]

[IBM Extension] Identifier-1 may also be defined as a NATIONAL item. The data accepted will be converted from the code set specified by the job's current CCSID. [End of IBM Extension]

[IBM Extension] Identifier-1 may not be a date, time, or timestamp item. [End of IBM Extension]

mnemonic-name
Must be specified in the SPECIAL-NAMES paragraph, where it will be associated with an environment-name that refers to an input device. The input device can be the workstation used by an interactive job, the job input stream of a batch job, or the system operator's console.

[IBM Extension]

environment-name
The environment-name CONSOLE or SYSIN may be specified in place of a mnemonic-name.

[End of IBM Extension]

Source of Input Data

The source of input data is dependent upon the type of program initiation as follows:

Method of Program Initiation	Data Source (Input Device)		
	When Associated with Environment-Name		When FROM Phrase Omitted
	CONSOLE or SYSTEM-CONSOLE	SYSIN or REQUESTOR	
BATCH	System operator's message queue	Job input stream	Job input stream
INTERACTIVE	System operator's message queue	Workstation	Workstation

The job input stream consists of data that accompanies a CL command. If there is no data in the input stream, or if there is insuffient data to fill identifier-1, an exception occurs.

When the input is from the `job input stream`, the following rules apply:

- An input record size of 80 characters is assumed.
- If identifier-1 is up to 80 characters in length, the input data must appear as the first characters within the input record. Any characters beyond the length of identifier-1 are truncated.
- If identifier-1 is longer than 80 characters, succeeding input records are read until the storage area of identifier-1 is filled. If the length of identifier-1 is not an exact multiple of 80 characters, the last input record is truncated.

When the device is the `workstation`, the input record size is 100. 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 workstation operator. Previous DISPLAYs can also appear on the ACCEPT screen.
2. Processing is suspended.
3. The reply is moved into identifier-1, 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 identifier-1 is longer than the input record size, succeeding input records are read (steps 1-3) until identifier-1 is filled.

If the incoming reply is longer than identifier-1, the character positions beyond the length of identifier-1 are truncated.

Note: If the device is the same as that used for READ statements, results are unpredictable.

Coding Example

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
```

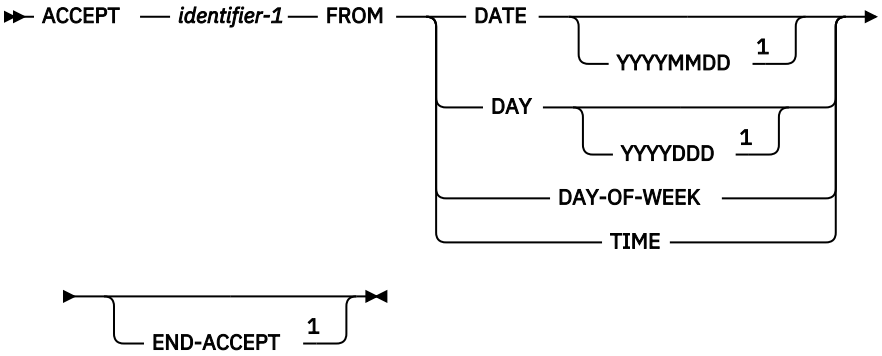
The following is an example of a COBOL program that uses a Format 1 ACCEPT statement to read the job input stream:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ACCPT1X.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-ISERIES.
OBJECT-COMPUTER. IBM-ISERIES.
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 2 - System Information Transfer

System information contained in the specified conceptual data items DATE, DAY, DAY-OF-WEEK, or TIME can be transferred into the identifier. DATE, DAY, DAY-OF-WEEK, 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 338.



ACCEPT Statement - Format 2 - System Info Transfer
Notes:

[1] IBM Extension

identifier-1
The receiving data item.

[IBM Extension] If the description of identifier-1 contains a TYPE clause, the type-name referenced in that clause must be elementary. [End of IBM Extension]

Format 2 accesses the current date 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.

Note: The current date and time are also accessible using the CURRENT-DATE Intrinsic Function (see “CURRENT-DATE” on page 475).

DATE, DAY, DAY-OF-WEEK, and TIME

The conceptual data items DATE, DAY, DAY-OF-WEEK, and TIME implicitly have USAGE DISPLAY.

DATE (Without the YYYYMMDD Phrase)
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, 16 November 1963 is expressed as:

```
631116
```

[IBM Extension]

DATE (With the YYYYMMDD Phrase)
Has the implicit PICTURE 9(8).

The sequence of data elements (from left to right) is:

```
4 digits for year in the Gregorian calendar
2 digits for month of year
2 digits for day of month
```

Thus, 16 November 1963 is expressed as:

```
19631116
```

[End of IBM Extension]

DAY (Without the YYYYDDD Phrase)
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
```

[IBM Extension]

DAY (With the YYYYDDD Phrase)
Has the implicit PICTURE 9(7).

The sequence of data elements (from left to right) is:

```
4 digits for year in the Gregorian calendar
3 digits for day of year
```

Thus, 31 December 1995 is expressed as:

```
1995365
```

[End of IBM Extension]

DAY-OF-WEEK
Has the implicit PICTURE 9(1).

The single data element represents the day of the week thus:

```
1 represents Monday
2 represents Tuesday
3 represents Wednesday
4 represents Thursday
5 represents Friday
```

```
6 represents Saturday
7 represents Sunday
```

Thus Thursday is expressed as: 4

TIME
Has the implicit PICTURE 9(8).

The sequence of data elements (from left to right) is:

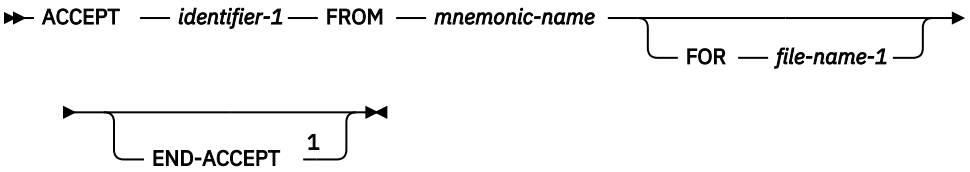
```
2 digits for hour of day
2 digits for minute of hour
2 digits for second of minute
2 digits for hundredths of second
```

Thus 12.25 seconds after 2:41 PM is expressed as:

```
14411225
```

[IBM Extension] Format 3 - Feedback

This format is used to transfer feedback information from an active file to the identifier.



ACCEPT Statement - Format 3 - Feedback
Notes:

[1] IBM Extension

Identifier-1 can be any fixed-length group item or an elementary alphabetic, alphanumeric, or external decimal item. Identifier-1 cannot be a date-time item. If the description of identifier-1 contains a TYPE clause, the type-name referenced in that clause must be elementary.

Identifier-1 can also be an internal or external floating-point data item.

File-name-1 must be defined in an FD entry, and must be open prior to the execution of the ACCEPT statement. If file-name-1 is not open, the contents of identifier-1 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.

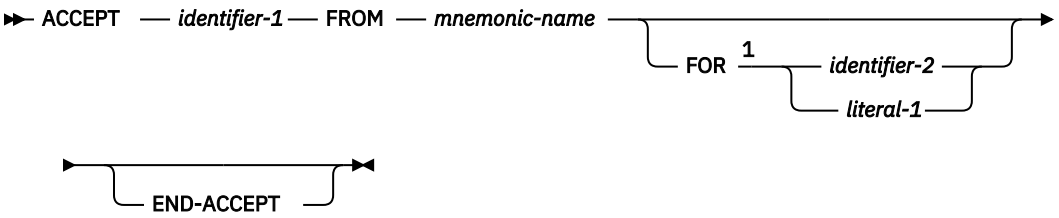
When the FOR phrase is specified, the feedback information is from the file specified in the phrase. When the FOR phrase is not specified, the feedback information is from the last file opened or used in an input or output operation of the current program.

See *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for a discussion of the I-O-FEEDBACK and OPEN-FEEDBACK areas. For a layout and description of the fields contained in the feedback areas, see the *Db2 for i* section of the *Database and File Systems* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

[End of IBM Extension]

[IBM Extension] Format 4 - Local Data Area

This format is used to transfer data to identifier-1 from the system-defined local data area created for a job.



ACCEPT Statement - Format 4 - Local Data Area

Notes:

[1] Syntax-checked only.

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. Identifier-1 cannot be a date-time item. If the description of identifier-1 contains a TYPE clause, the type-name referenced in that clause must be elementary.

Identifier-1 can be an internal or external floating-point data item.

Identifier-1 can be a DBCS or national data item.

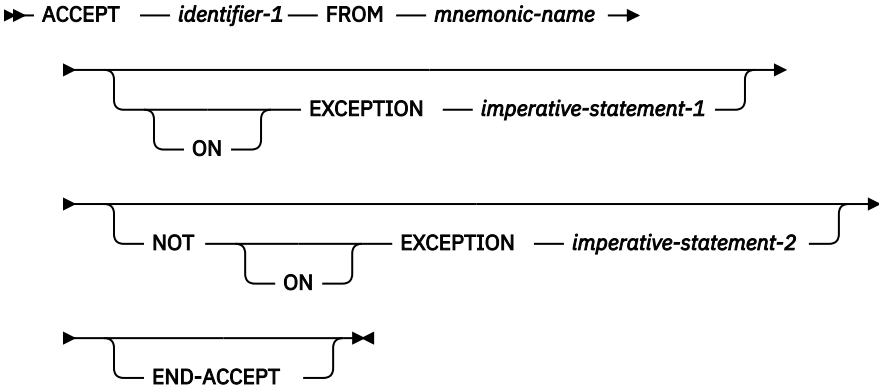
When the FOR phrase is specified, it is syntax checked during compilation but treated as a comment during execution. The value of literal-1 or identifier-2 indicates the program device name of the device that is associated with the local data area. There is only one local data area for each job, and all devices in a job access the same local data area. Literal-1, 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 Programming* manual.

[End of IBM Extension]

[IBM Extension] Format 5 - Program Initialization Parameters

You use this format to transfer data from the PIP (Program Initialization Parameters) data area into the identifier.

ACCEPT Statement - 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 identifier-1 takes place according to the rules for the MOVE statement for a group move without the CORRESPONDING phrase. Identifier-1 cannot be a date-time item. If the description of identifier-1 contains a TYPE clause, the type-name referenced in that clause must be elementary.

Identifier-1 can be an internal or external floating-point data item.

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. 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 243.

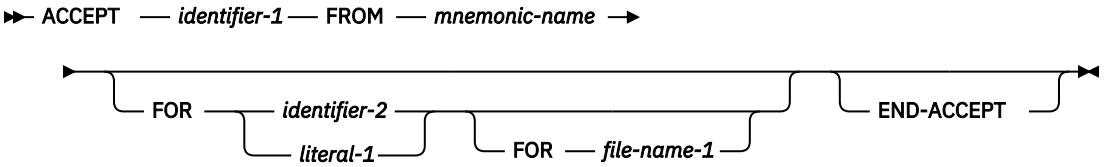
Note that you cannot update the PIP data area using COBOL. For more information about the PIP data area, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* and the *CL Programming* book.

[End of IBM Extension]

[IBM Extension] Format 6 - Attribute Data

The ACCEPT statement retrieves information (attribute data) about a particular program device associated with a TRANSACTION file.

ACCEPT Statement - Format 6 - Attribute Data



This format of the ACCEPT statement may only be used for files with an organization of TRANSACTION. Identifier-1 cannot be a date-time item. If the description of identifier-1 contains a TYPE clause, the type-name referenced in that clause must be elementary.

Identifier-1 can be an internal or external floating-point data item.

Identifier-1 can be a DBCS or national data item.

If file-name-1 is not open at the time the ACCEPT is executed, message LNR7205 is issued.

Mnemonic-name must be associated with the environment-name ATTRIBUTE-DATA in the SPECIAL-NAMES paragraph.

If file-name-1 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-1, 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 LNR7205 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 Programming* manual 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.

[End of IBM Extension]

Attribute Data Formats

The attribute data retrieved by the ACCEPT statement depends on whether the data and the associated fields are applicable to a workstation or to a communications device. See "Appendix F. File Structure Support Summary and Status Key Values" on page 563 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.

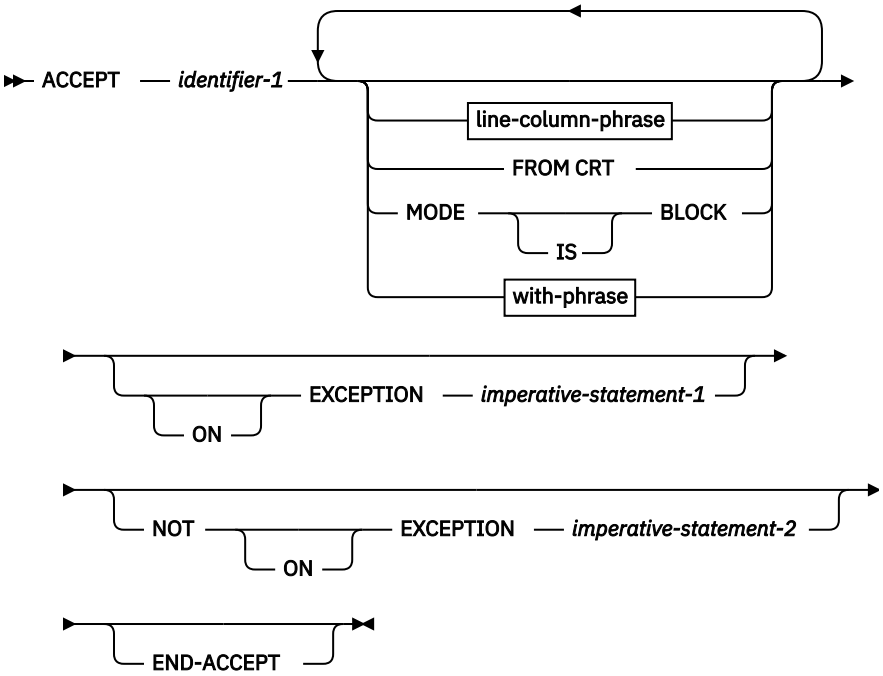
[IBM Extension] Workstation I/O

An ACCEPT statement is considered an **extended** ACCEPT statement if it:

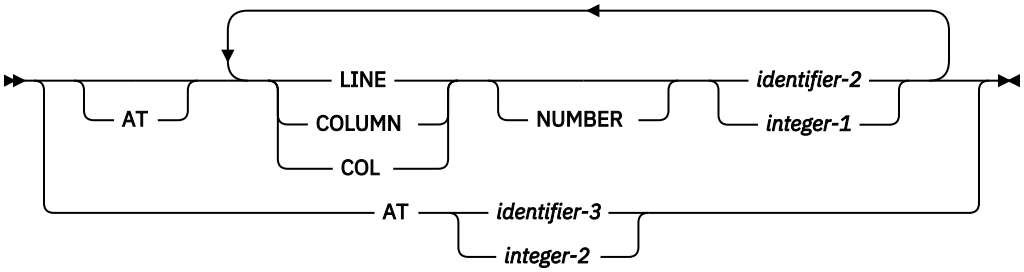
- has an AT phrase, or
- has a FROM phrase with the CRT option, or
- has a MODE IS BLOCK phrase, or
- has a WITH phrase, or
- has an ON EXCEPTION phrase or a NOT ON EXCEPTION phrase, (and PIP-DATA is not specified for mnemonic-name), or
- does not have a FROM phrase, but CONSOLE IS CRT is specified in the SPECIAL-NAMES paragraph.

An ACCEPT statement is considered a **standard** ACCEPT statement if it:

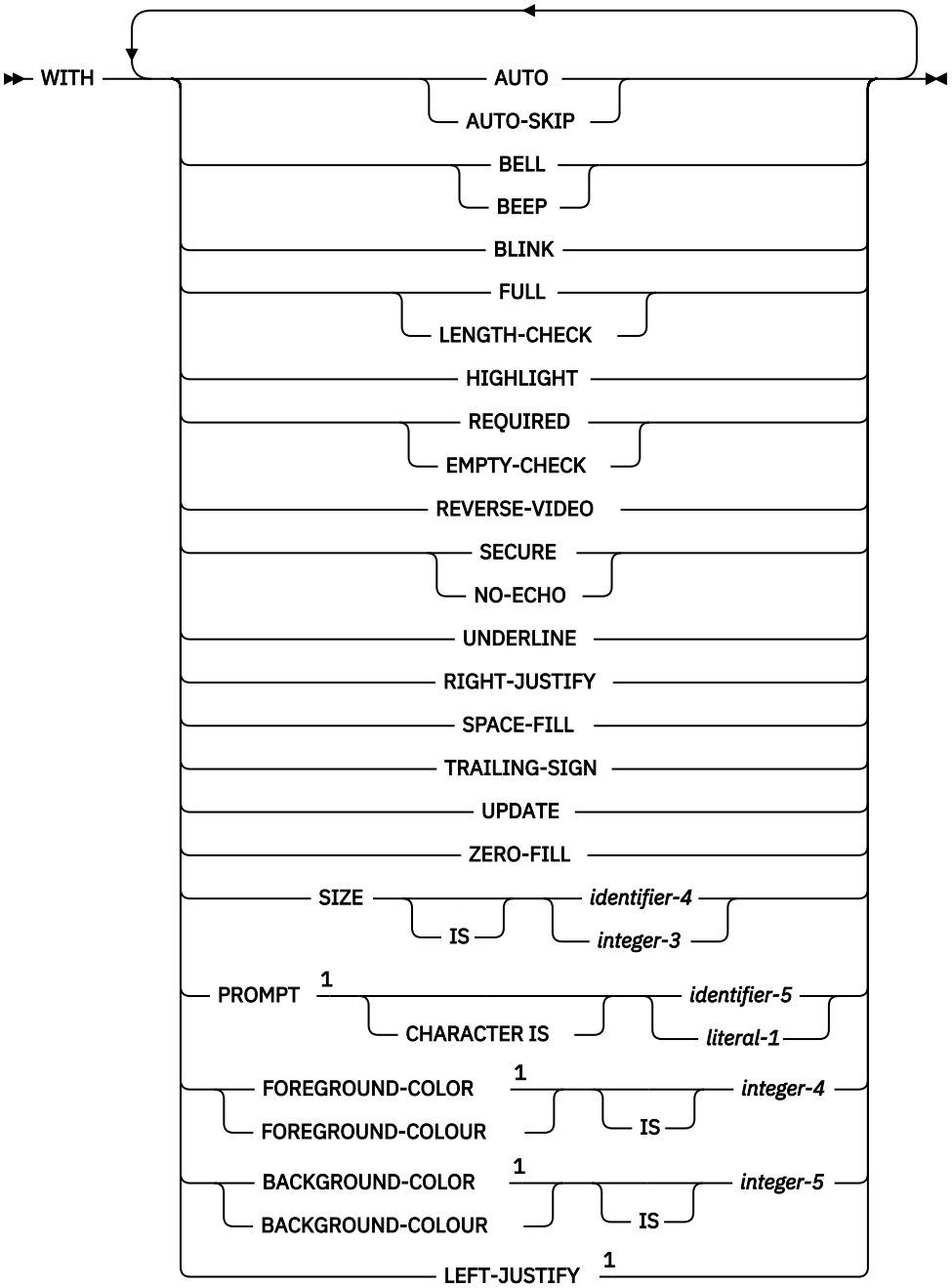
- has a FROM phrase (other than FROM CRT) and CONSOLE IS CRT is specified in the SPECIAL-NAMES paragraph, or
- does not have a FROM phrase and CONSOLE IS CRT is **not** specified.



line-column-phrase



with-phrase



ACCEPT Statement - Format 7 - Workstation I/O

Notes:

[1] Syntax-checked only.

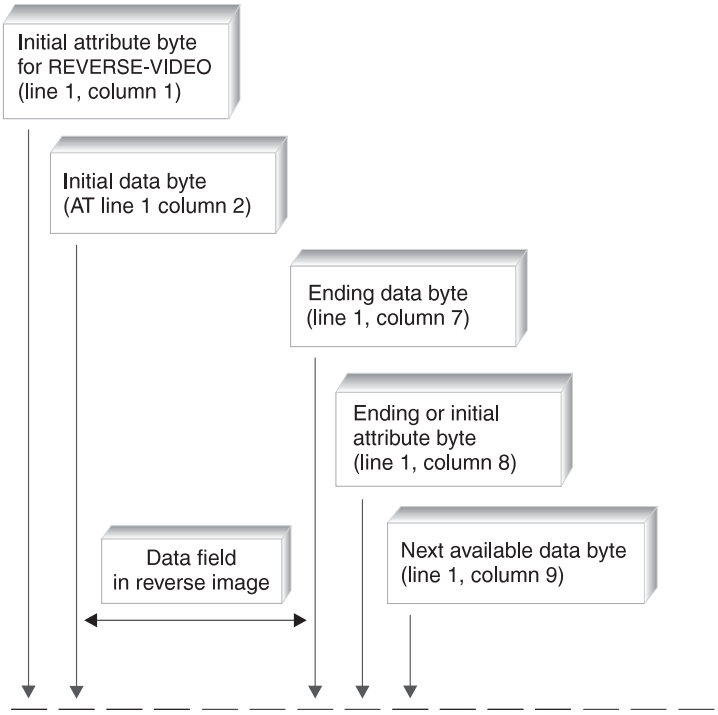
identifier-1

A data item whose value can be updated.

Identifier-1 can be an internal or external floating-point data item.

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 your 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. You should also be aware that the ending attribute byte will be the normal attribute defined for the specific workstation. Therefore, you should ensure that the attributes are specified in the correct order to obtain the expected results.

You should initially clear the screen by using a DISPLAY statement that contains the WITH BLANK SCREEN phrase.

When identifier-1 does not fit within the screen, then alphanumeric data is truncated and numeric data is not put on the screen.

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.

REDEFINES, POINTER, PROCEDURE-POINTER, and INDEX-NAME data are ignored when found in the group item without the MODE IS BLOCK phrase.

The extended ACCEPT statement predisplays the value of identifier-1, accepting only the changed value from the workstation after the ENTER key is pressed. Simply pressing the ENTER key does not cause an update of the predisplayed data item.

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.

The FIELD EXIT key will convert all trailing spaces in the alphabetic and alphanumeric fields to hexadecimal zeroes when the default compiler option *UNDSPCHR is in effect. For example:

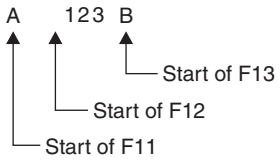
```
DATA DIVISION.
       01    STRUC1.
             03  F11 PIC AA VALUE 'A'.
             03  F12 PIC 9(4) VALUE 123.
             03  F13 PIC XXX VALUE 'B'.
PROCEDURE DIVISION.
       ACCEPT STRUC1 AT 2102.
```

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='B␣␣' 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, internal floating-point, or packed data (directly or redefined).
- Extended ACCEPT statements do not use binary, internal floating-point, 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 nondisplayable characters because it will be handled as one alphanumeric field 13 bytes long.

When the program runs on a workstation with a remote controller and with 5250 emulation, the ILE COBOL run time changes compiler option *UNDSPCHR, if it is in effect, to *NOUNDSDPCHR and sends an informational message to the user. When your system configuration encompasses a variety of workstation controllers, use of the *NOUNSDPCHR option is recommended to achieve consistency of results. To enforce this option effectively, specify NOUNDSPCHR on the PROCESS statement in the COBOL source program.

[End of IBM Extension]

Considerations for Floating-Point Data Items

You should consider the following when using floating-point data items with the ACCEPT statement.

It is possible that when an external floating-point literal is ACCEPTed, slight inaccuracies can result. This is especially true if the floating-point data item is moved after it is ACCEPTed. The floating-point data type is an approximation, and when an external floating-point literal is moved, it is first converted to a true floating-point value (IEEE), which can also affect its accuracy.

For example, consider the following ACCEPT:

```
77 external-float-1 PIC +9(3).9(13)E+9(3).
    ACCEPT external-float-1 FROM CRT.
    DISPLAY "EXTERNAL-FLOAT-1=" external-float-1.
```

The displayed result after +123455779012.3453E+297 is ACCEPTed is:

```
EXTERNAL-FLOAT-1=+123.4557790123452E+306
```

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. (The extended ACCEPT and DISPLAY statements do not support data items with scaling positions in the PICTURE clause.)

Table 31. Data Categories Handled by Extended ACCEPT

Category	Initial Display	Entering Data	Data Item Updated
Alphabetic	A, B, C	D	B, C
Numeric (internal, binary, 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
DBCS	A, B, C	D	B, C
DBCS-edited	A, B, C	D	B, C, L
Internal floating-point	A, I, I1, P	Q	O
External floating-point	A, I, I1	Q	O

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 workstation will right-justify and zero-fill or space-fill the field when the field exit key is pressed. ZERO-FILL does not work with DBCS.

If only ZERO-FILL (or SPACE-FILL) is specified, the workstation 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 implicit or explicit PIC symbol. ZERO-FILL, SPACE-FILL, and RIGHT-JUSTIFY do not affect an edited or floating-point field.

I1

TRAILING-SIGN does not affect an edited or floating-point 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 131 for the rules about positioning data.

P

An internal floating-point number is converted to external floating-point before it is displayed.

- A COMP-1 item will display as if it had an external floating-point PICTURE clause of -.9(8)E-99
- A COMP-2 item will display as if it had an external floating-point PICTURE clause of -.9(17)E-999

Q

Data must be entered following the rules for formation of a floating-point literal (see “Floating-Point Literals” on page 37). The exponent is optional.

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.

COL is an abbreviation for COLUMN.

The LINE and COLUMN phrases can appear in any order.

identifier-2, integer-1

Identifier-2 and integer-1 must be an unsigned numeric integer with a value greater than or equal to zero and less than 9 digits. 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.

Identifier-2 cannot be an internal or external floating-point data item.

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. Identifier-3 cannot be an internal or external floating-point data item.

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 workstation 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.

SIZE Phrase

Specifies the size of the data item on the screen. You can use this phrase with elementary data items only.

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.

When the size specified for a numeric or numeric-edited data item is less than that implied by the PICTURE clause, truncation of the rightmost positions occurs when the value is displayed, or predisplayed in the ACCEPT operation. The data item is then updated following the rules for the MOVE operation.

If you specify a SIZE literal whose value causes the field length to exceed the screen size, alphanumeric data will be truncated and numeric data will be ignored and not displayed.

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.

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

Syntax checked and then treated as documentation by the compiler.

- **PROMPT CHARACTER**
  - The PROMPT CHARACTER clause causes the empty character positions on the screen to be marked.

    identifier-5
    Identifier-5 must be a single-character alphabetic or alphanumeric data item. Identifier-5 must not be subject to an OCCURS clause.

    Literal-1
    Literal-1 must be a 1-character nonnumeric literal or a figurative constant.
- **FOREGROUND-COLOR** or **FOREGROUND-COLOUR**
  - The FOREGROUND-COLOR/FOREGROUND-COLOUR clause specifies the foreground color of the screen item.

    integer-4
    Integer-4 must be an unsigned numeric integer.
- **BACKGROUND-COLOR** or **BACKGROUND-COLOUR**
  - The BACKGROUND-COLOR/BACKGROUND-COLOUR clause specifies the background color of the screen item.

    integer-5
    Integer-5 must be an unsigned numeric integer.
- **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.

Extended ACCEPT and Extended DISPLAY Considerations

The following considerations are common to both the extended ACCEPT and the extended DISPLAY statements.

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.

Subscripting and Reference Modification

Subscripted items, and reference modified items are both supported.

Performance

Unless you specify the EXTDSPOPT(*NODFRWRT) parameter (no deferred writing) in the CRTCBLMOD or CRTBNDCBL command, the ILE COBOL 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 *IBM Rational Development Studio for i: ILE COBOL Programmer'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 *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.
- Unless the user specifies the *NOUNDSPCHAR (no undisplayable characters) option of the extended display parameter of the CRTCBLMOD or CRTBNDCBL command, or the equivalent process statement option, data is passed to the screen exactly as sent. If the *NOUNDSPCHAR option is specified, the data is examined by the workstation for the presence of control information. In that case, output data must contain only valid displayable characters.
- 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 (LNC0265) 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.

TRANSACTION Files

Using extended ACCEPT/DISPLAY statements and TRANSACTION files in the same program is not recommended. If extended ACCEPT/DISPLAY statements are used in the same program as TRANSACTION files, then the TRANSACTION file should be closed when the extended ACCEPT/DISPLAY statements are performed. Unpredictable results will occur if an extended ACCEPT/DISPLAY statement is

performed when a TRANSACTION file is open. A severe error may be generated or data on the workstation may be overlapped or intermixed.

Remote Workstations

Extended ACCEPT and extended DISPLAY statements do not run on remote workstations attached to 5251 Model 12 controllers.

The EXTDSPOPT(*NOUNDSPCHR) parameter in the CRTCBLMOD or CRTBNDCBL command 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. The CLEAR and HELP keys cannot be used to accept data when using remote controllers.

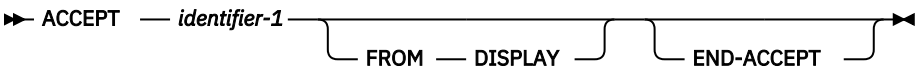
Differences from COBOL/2 Processing*

The ILE COBOL extended ACCEPT and DISPLAY statements are similar to the ACCEPT and DISPLAY statements (Format 2). The exceptions are discussed in "Appendix I. ACCEPT/DISPLAY and COBOL/2 Considerations" on page 588.

[IBM Extension] Format 8 - Session I/O

The ACCEPT statement retrieves information from the ILE common session manager.

ACCEPT Statement - Format 8 - Session I/O



If the description of identifier-1 contains a TYPE clause, the type-name referenced in that clause must be elementary.

For this format of the ACCEPT statement the FROM phrase is optional if the CONSOLE IS DISPLAY clause is specified in the SPECIAL-NAMES paragraph.

Format 8 transfers data from the ILE common session manager into identifier-1. The incoming data is received in one of the following formats:

- USAGE IS DISPLAY format. The data is not converted.
- USAGE IS DISPLAY-1 format. The shift-out and shift-in characters that surround the data are stripped off.
- USAGE IS NATIONAL format, The data is converted from the code set specified by the job's current CCSID.

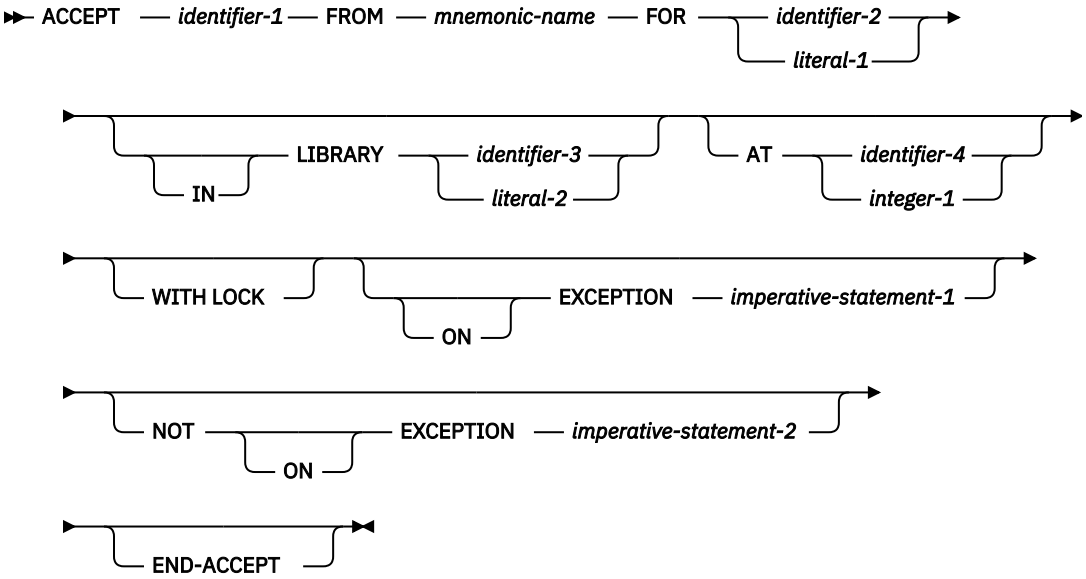
The ILE common session manager is used to manage the ACCEPT statement. For further information on the screen I/O session services, refer to the "Dynamic Screen Manager" section in the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

[End of IBM Extension]

[IBM Extension] Format 9 - Data Area

The ACCEPT statement retrieves information from the data area specified in the FOR phrase.

ACCEPT Statement - Format 9 - Data Area



For more information and an example of how to use data areas, see the "Using Data Areas You Create" information in the "Passing Data Using Data Areas" section of the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

[End of IBM Extension]

identifier-1

Identifier-1 can be class alphanumeric, numeric, DBCS, or national.

If the description of identifier-1 contains a TYPE clause, the type-name referenced in that clause must be elementary.

FROM Phrase

The FROM phrase specifies a mnemonic-name that must be associated with an environment name of DATA-AREA in the SPECIAL NAMES paragraph.

FOR Phrase

Identifies the operating system data area to retrieve information from. If the specified data area cannot be found at runtime, an ON EXCEPTION error occurs.

identifier-2
Must be an alphanumeric data item. The contents of identifier-2 must represent a valid operating system data area name. Operating system data area names are at most 10 characters long, thus the first 10 characters of identifier-2 are used to form the data area name.

literal-1
Must be nonnumeric and at most 10 characters long.

LIBRARY Phrase

Is used to specify the name of the operating system library in which the data area is to be found. The special values *LIBL (search using the job's library list) or *CURLIB (search the current library) may be specified. If the LIBRARY phrase is omitted, the job's library list is used to search for the data area.

identifier-3
Must be an alphanumeric data item. Since IBM i library names are at most ten characters long , only the first ten characters of identifier-3 are used to form the library name.

literal-2
Must be nonnumeric and at most 10 characters long.

Identifier-2, identifier-3, literal-1, and literal-2 are *not* affected by the *MONOPRC compiler option. They can contain an operating system quoted name (for details, see "Rules for Specifying Names" in the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/).

AT Phrase

The AT phrase indicates the starting position in the data area from which text is received.

If the AT phrase is not specified, a starting position of 1 is assumed.

identifier-4, integer-1
Identifier-4 and integer-1 must be positive numeric integers with a value that ranges from 1 to the maximum data area size (2000).

WITH LOCK Phrase

While the ACCEPT statement is retrieving information into identifier-1, the data area is locked with an LSRD (Lock Shared for Read) lock to prevent the data area from being changed. After identifier-1 is accepted, the WITH LOCK phrase places a LEAR (Lock Exclusive Allow Read) lock on the data area. If a lock cannot be placed on the data area, an exception condition occurs.

To maintain a lock on the data area after the transfer of data, specify this phrase. If a lock existed on the data area prior to this statement and the statement did not contain a WITH LOCK phrase, the lock is released.

(NOT) ON EXCEPTION Phrase

If an error occurs while accessing the data-area, then 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 data area is accessed successfully, any imperative statement specified in the NOT ON EXCEPTION phrase is processed.

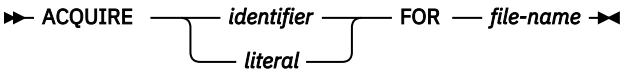
END-ACCEPT Phrase

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 the section on “Delimited Scope Statements” on page 243.

[IBM Extension] ACQUIRE Statement

The ACQUIRE statement acquires a program device for a TRANSACTION file.

ACQUIRE Statement - 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. The literal must be nonnumeric and 10 characters or less in length. The identifier 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 communication device can be acquired, see the *ICF Programming* manual. For a description of conditions that must be met before a display station can be acquired, refer to the *Db2 for i* section of the *Database and File Systems* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

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.

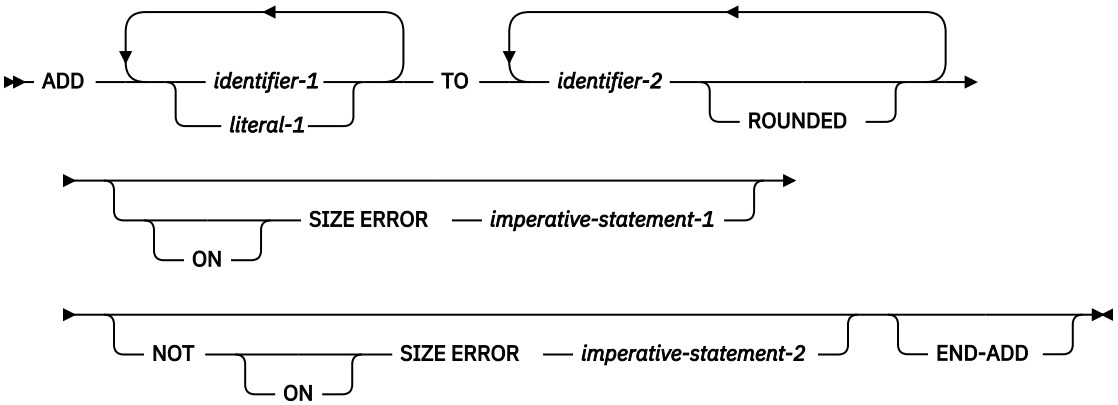
For more information on these CL commands for display stations, see the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/. See the *ICF Programming* manual for information on these CL commands for communication 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 *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

[End of IBM Extension]

ADD Statement

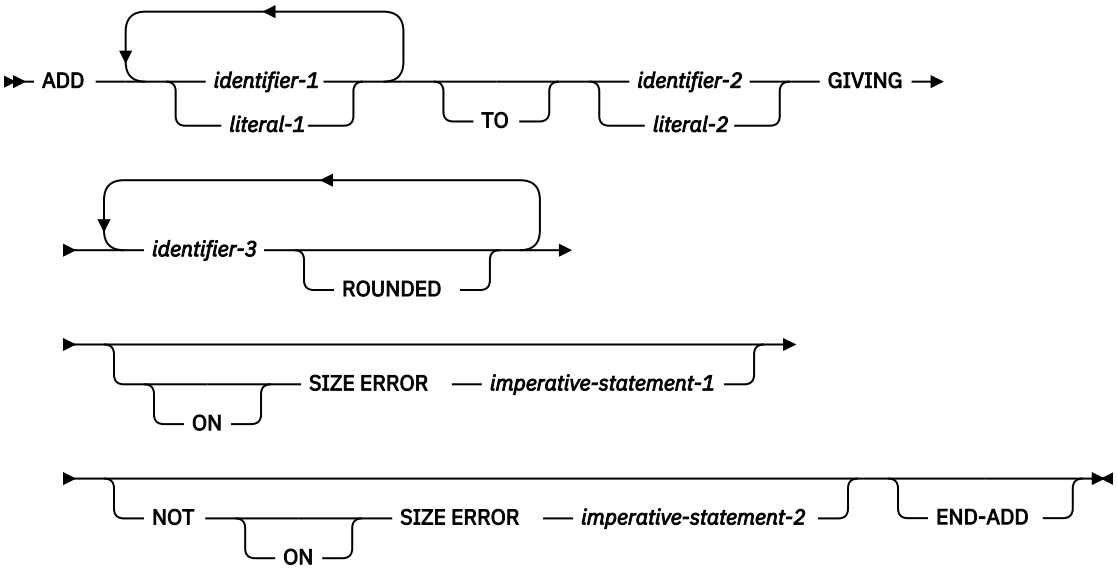
The ADD statement adds two or more numeric operands and stores the result.

ADD Statement - Format 1 - ADD



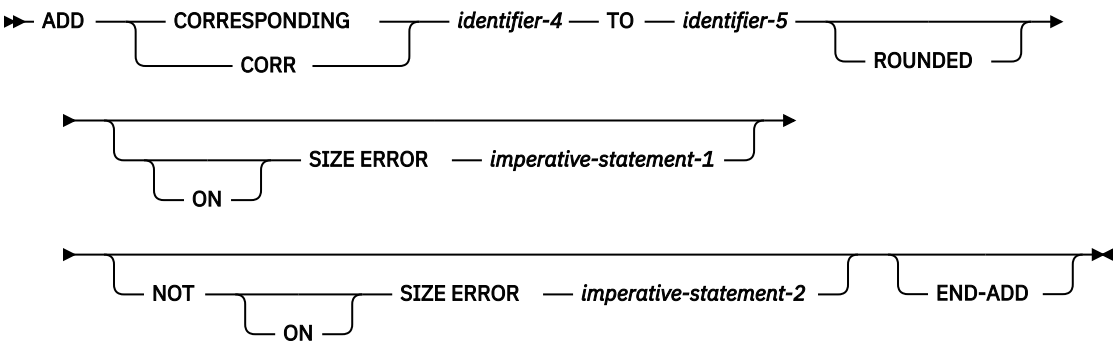
In Format 1, identifiers and literals preceding the keyword TO are added together, and this initial sum is added to and stored in identifier-2. The initial sum is also added to each successive occurrence of identifier-2, in the left-to-right order in which identifier-2 is specified.

ADD Statement - Format 2 - ADD GIVING



In 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.

ADD Statement - Format 3 - ADD CORRESPONDING



In Format 3, elementary data items within identifier-4 are added to and stored in the corresponding elementary items within identifier-5.

For all Formats:

identifier-1, identifier-2
Must be an elementary numeric item.

identifier-3
Must be an elementary numeric item or a numeric-edited item.

identifier-4, identifier-5
Must be a group item.

literal-1, literal-2
Must be a numeric literal.

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.

For more information on the composite of operands, see the "Size of Operands" on page 247.

[IBM Extension] Floating-point data items and literals can be used anywhere a numeric data item or literal can be specified. [End of IBM Extension]

ROUNDED Phrase

See "ROUNDED Phrase" on page 246.

SIZE ERROR Phrases

See "SIZE ERROR Phrases" on page 246.

CORRESPONDING Phrase (Format 3)

See "CORRESPONDING Phrase" on page 245.

END-ADD Phrase

This explicit scope terminator delimits the scope of the ADD statement. END-ADD converts a conditional ADD statement into an imperative statement so that it can be nested in another conditional statement.

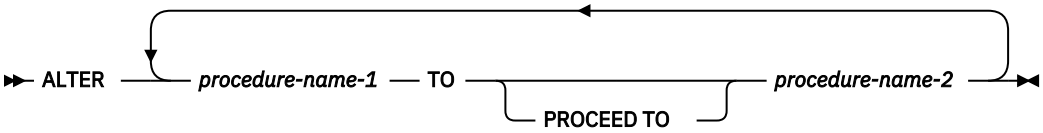
For more information, see "Delimited Scope Statements" on page 243.

ALTER Statement

The ALTER statement changes the transfer point specified in a GO TO statement.

Note: The ALTER statement encourages the use of unstructured programming practices. The EVALUATE statement provides the same function as the ALTER statement and helps to ensure that your program will be well-structured.

ALTER Statement - 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 be a Procedure Division paragraph that contains only one sentence: a GO TO statement without the DEPENDING ON phrase.

procedure-name-2
Must be a Procedure Division section or paragraph.

If procedure-name-1 or procedure-name-2 are within a declarative procedure, neither can reference any nondeclarative procedure. 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 procedures associated with it.

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.

Note: Do not use the ALTER statement in programs that have the RECURSIVE attribute.

Coding Example

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.

Altered GO TO statements in programs with the INITIAL attribute are returned to their initial state each time the program is entered.

CALL Statement

The CALL statement transfers control from one 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.

[IBM Extension] In ILE COBOL, a subprogram may be a COBOL program, a program written in another IBM i language, or an ILE procedure. [End of IBM Extension]

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. If the called subprogram has no procedure division or nondeclarative section in the Procedure Division, the called subprogram issues an implicit EXIT PROGRAM.

Whenever program control is transferred by the CALL statement and the called program directly or indirectly executes its caller, a **recursive** call has been made. Programs defined with the RECURSIVE attribute can execute a CALL statement that directly or indirectly executes itself. ILE COBOL does not allow recursion in non-recursive programs. A run time error message will be generated if recursion is attempted for a non-recursive program. For more information on calling programs and the associated concepts and terminology, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

The RETURN-CODE special register can be used to pass return code information from a program to its caller. See “RETURN-CODE Special Register” on page 415 for further information.

CALL statement processing passes control to the called subprogram. If a CALL statement has a linkage of program object and the CALL statement names a program that does not exist in the specified library at run time, an error message is issued. The ON EXCEPTION or OVERFLOW phrase can be used to specify an error handling procedure.

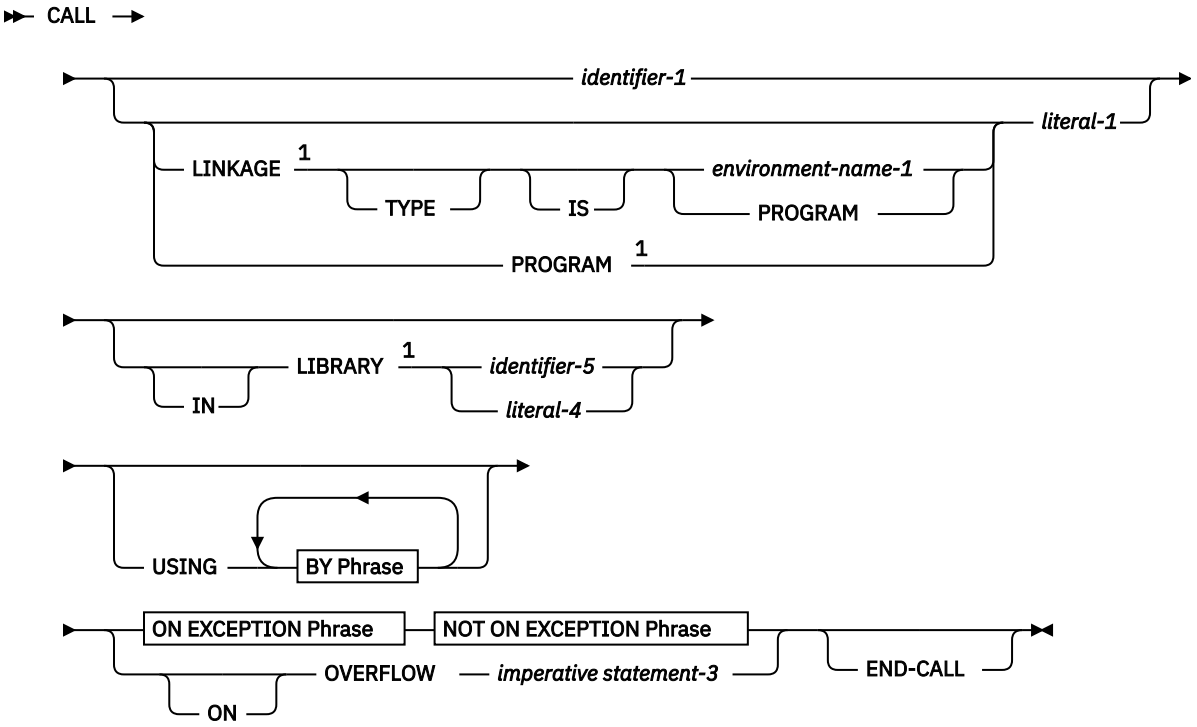
A called subprogram is in its initial state the first time it is called within a run unit. It is also in its initial state the first time it is called after a CANCEL statement.

A program is in its initial state each time it is called if it is an initial program (if its PROGRAM-ID paragraph contains the INITIAL clause). On all other entries into the called subprogram, the subprogram is in its last-used state, except in the case of the PERFORM statement control mechanisms; these are always set to their initial state.

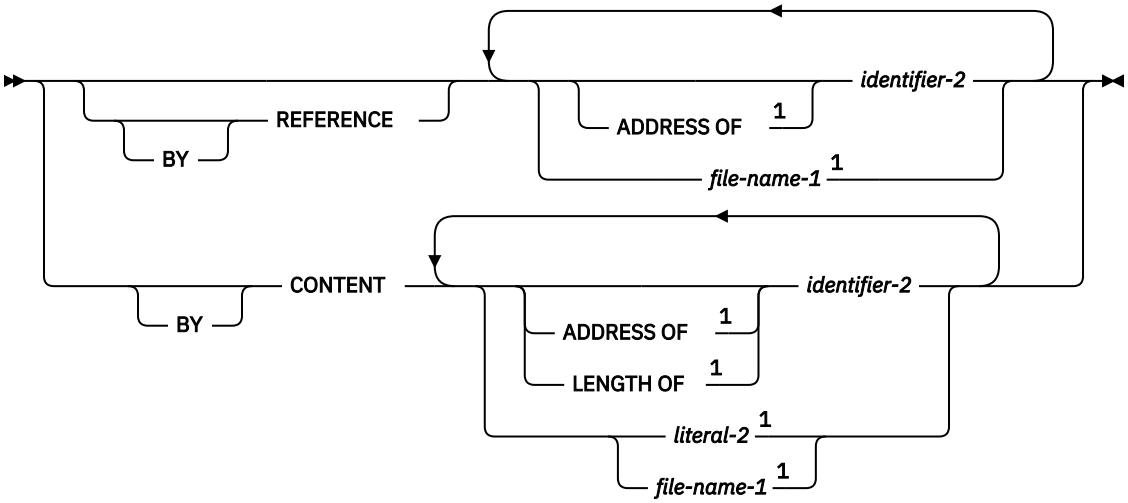
Whenever an implicit or explicit STOP RUN occurs, the Languages and Utilities return code is set to 0. Otherwise, it is set to 2. The RETURN-CODE register is copied to the user portion of the work control

block. See the RTVJOBA and DSPJOB commands in the *CL Programming* book for more information about return codes.

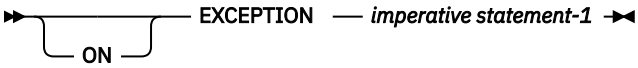
The user return code is set to 0 at the start of the processing of any COBOL program, and before a call is made to another program.



BY Phrase



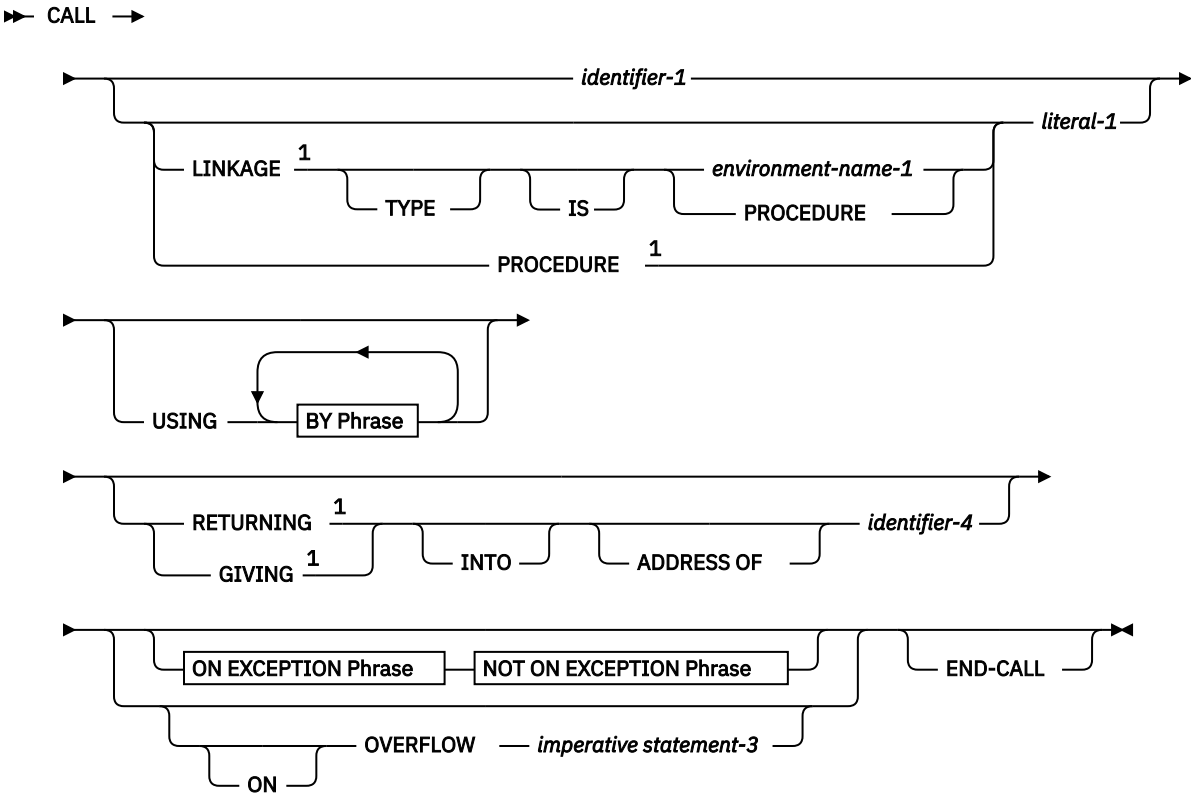
ON EXCEPTION Phrase



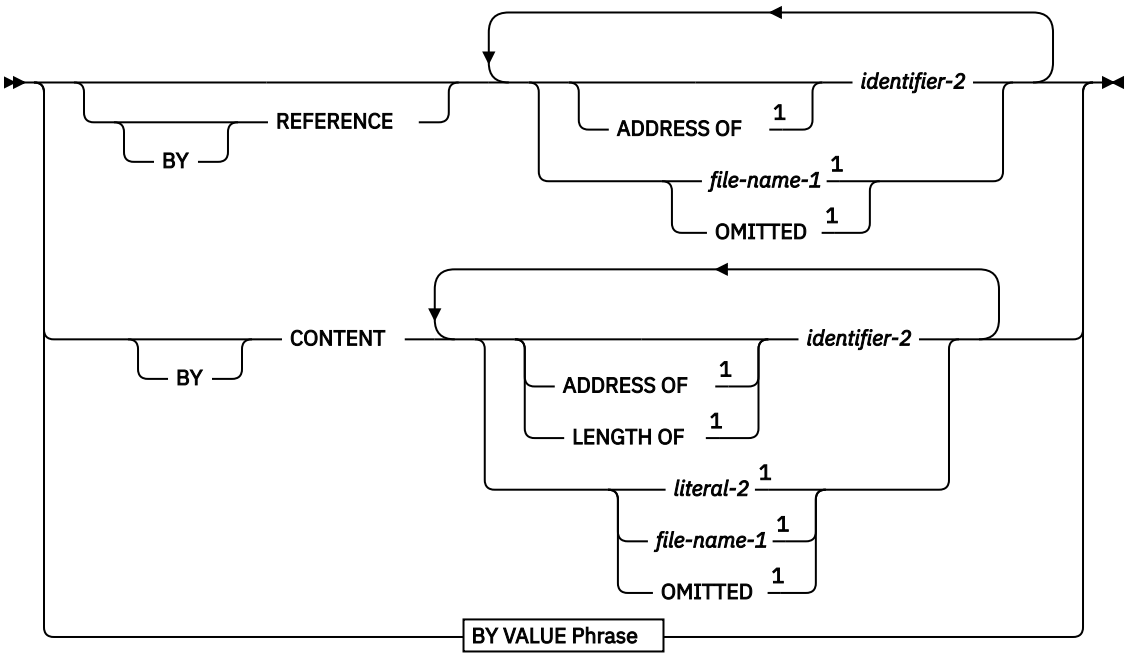
NOT ON EXCEPTION Phrase

CALL Statement - Format 1
Notes:

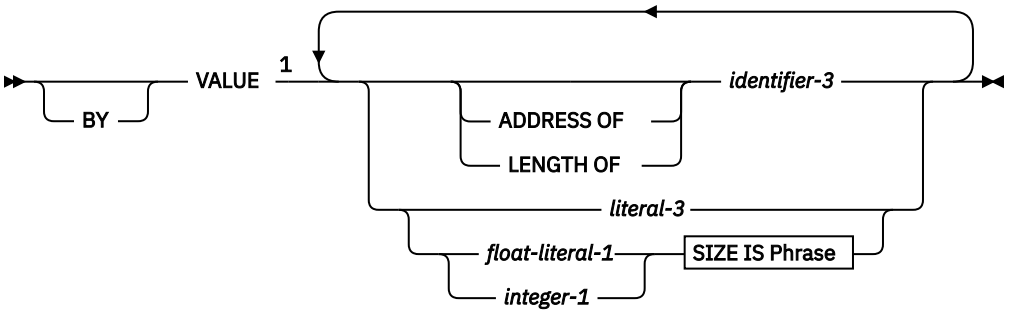
[1] IBM Extension



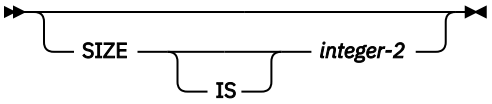
BY Phrase



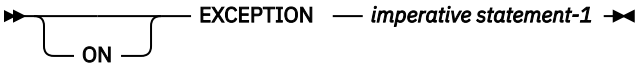
BY VALUE Phrase



SIZE IS Phrase



ON EXCEPTION Phrase



NOT ON EXCEPTION Phrase

CALL Statement - Format 2

Notes:

[1] IBM Extension

identifier-1

Must be an alphanumeric or a procedure-pointer data item.

For an alphanumeric data item, the following rules apply:

- If the linkage is to a program object, the contents of identifier-1 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 the linkage is to a procedure, the first 256 characters of identifier-1 are used. The called procedure must be in the same compilation unit as the calling procedure.
- Depending on the compiler option *MONOPRC, the contents of identifier-1 might need to be uppercase and conform to the rules for formation of program-names.

Procedure-pointer data items must be set to the address of a program or a procedure using the SET statement prior to the CALL. For information on setting procedure-pointer data items, see “Format 6 - Procedure-Pointer Data Item” on page 399.

literal-1

The CALL linkage determines the type of program called and thus also restricts the content and size of literal-1. The linkage made is either to a program object or an ILE procedure. If the linkage is to a program object, then literal-1 must be nonnumeric, uppercase (except for an extended system name), 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. Literal-1 can contain an extended name.

If the linkage is to an ILE procedure, then literal-1 must be nonnumeric and at most 256 characters long. Depending on the compiler option *MONOPRC, literal-1 might need to be uppercase and conform to the rules for formation of program-names. The literal must specify the program-name of the called subprogram.

[IBM Extension] LINKAGE TYPE Phrase

The LINKAGE TYPE phrase is used to specify the type of linkage to be made on the CALL to literal-1.

environment-name-1
The type of linkage the compiler will generate for the CALL. Environment-name-1 can be defined as:

PGM
Linkage to a program object (*PGM) is generated.

PRC
Linkage to an ILE procedure is generated.

PROGRAM
Linkage to a program object (*PGM)

PROCEDURE
Linkage to an ILE procedure

If the LINKAGE TYPE phrase is not specified on the CALL statement, the linkage generated for the CALL can be changed by specifying one of: the LINKAGE TYPE clause of the SPECIAL-NAMES paragraph, or the LINKLIT parameter of the CRTCBLMOD or CRTBNDCBL command.

[End of IBM Extension]

[IBM Extension] IN LIBRARY Phrase

The LIBRARY phrase allows you to qualify IBM i program objects with an IBM i library name. If the LIBRARY phrase is not specified, the program object is searched for using the job's library list (*LIBL).

identifier-5
Must be an alphanumeric data item. The contents of identifier-5 must represent a valid IBM i library name. IBM i library names can be a maximum of 10 characters long. The first 10 characters of identifier-5 are used to form the library name.

literal-4
Must be nonnumeric and can be a maximum of 10 characters.

Identifier-5 and literal-4 are *not* affected by the *MONOPRC compiler option, and may contain an IBM i extended name.

[End of IBM Extension]

USING Phrase

Included in the CALL statement when parameters need to be passed to the called subprogram. If this is also written in COBOL, it must contain a USING phrase in its Procedure Division header. Procedure Division header of the called subprogram. The number of operands in both USING phrases must be identical. For CALL statements with a LINKAGE TYPE of program the maximum number of operands is 255, and for LINKAGE TYPE of procedure the maximum number of operands is 400.

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. For more information about the USING phrase, see "The USING Phrase" on page 219.

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.

[IBM Extension] Some procedures (for example, the ILE CEEDATE and CEEDAYS APIs) require that the **operational descriptor** of one or more parameters is made available. This requirement must be satisfied by including, in the SPECIAL-NAMES paragraph, a LINKAGE TYPE clause for the procedure with a USING phrase that specifies the appropriate parameter. In addition, any such parameter must be defined as an elementary data item with a USAGE of DISPLAY or DISPLAY-1, and it may not be reference modified. [End of IBM Extension]

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.

USING Phrase Example

Calling Program Description (PGMA)

```
WORKING-STORAGE
SECTION.
01 ARG-LIST.
   05 PARTCODE PIC A.
   05 PARTNO PIC X(4).
   05 U-SALES PIC 9(5).
   .
   .
   .
PROCEDURE DIVISION.
   .
   .
   .
      CALL PGMB
       USING ARG-LIST.
```

Called Program Description (PGMB)

```
LINKAGE SECTION.
01 PARAM-LIST.
    10 PART-ID PIC X(5).
    10 SALES PIC 9(5).
   .
   .
   .
PROCEDURE DIVISION USING
PARAM-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.

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 an ILE COBOL 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, Working-Storage, Local-Storage or Linkage Sections. Must not be a function-identifier.

[IBM Extension] It can be a:

- Data item of any level in the Data Division
- Pointer data item (an item defined implicitly or explicitly as USAGE IS POINTER)
- Procedure-pointer data item
- DBCS data item
- National data item
- Floating-point data item
- Date-time data item.

[End of IBM Extension]

[IBM Extension]

ADDRESS OF special register
For information about this special register, see page “ADDRESS OF Special Register” on page 126. Note that the calculated ADDRESS OF is not allowed in this case.

file-name-1
Must appear in an FD entry. It passes a null pointer data item.

OMITTED
For standard parameters when a parameter is passed BY REFERENCE, a pointer to the original data item is passed to the called program. When OMITTED is specified, a NULL pointer is passed to the called program. In this case, the called program will use its default value.

OMITTED can only be specified on calls to programs with a LINKAGE TYPE of procedure.

[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. This value is assigned to the corresponding parameter of the called program.

For each ILE COBOL 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, Working-Storage, Local-Storage or Linkage Sections. It must not be a function-identifier.

[IBM Extension] It can be a:

- Data item of any level in the Data Division
- Pointer data item (an item defined implicitly or explicitly as USAGE IS POINTER)
- Procedure-pointer data item
- DBCS data item
- National data item
- Floating-point data item
- Date-time data item.

[End of IBM Extension]

[IBM Extension]

ADDRESS OF special register
For information about this special register, see page “ADDRESS OF Special Register” on page 126.

ADDRESS OF a data item
For information about this, see page “ADDRESS OF” on page 126.

LENGTH OF special register
The LENGTH OF special register contains the number of bytes used by a data item referenced by an identifier. For more information, see “LENGTH OF Special Register” on page 286.

literal-2
Can be:

- A nonnumeric literal
- A figurative constant
- A Boolean literal
- A DBCS literal
- National literal.

file-name-1
Must appear in an FD entry. It passes a pointer data item.

OMITTED
For standard parameters when a parameter is passed BY CONTENT, a pointer to a copy of the data item is passed to the called program. When OMITTED is specified, a NULL pointer is passed to the called program. In this case, the called program will use its default value.

OMITTED can only be specified on calls to programs with a LINKAGE TYPE of procedure.

[End of IBM Extension]

[IBM Extension] BY VALUE Phrase

When the BY VALUE phrase is specified, the value of the parameter is passed, not a reference to the sending data item. The called program can modify the formal parameter corresponding to the BY VALUE parameter, but any such changes do not affect the parameter since the called program has access to a temporary copy of the sending data item.

While BY VALUE parameters are primarily intended for communication with non-COBOL programs (such as C), they can also be used for COBOL-to-COBOL invocations. In this case, BY VALUE must be specified or implied for both the parameter in the CALL USING phrase and the corresponding formal parameter in the PROCEDURE DIVISION USING phrase.

The BY CONTENT, BY VALUE and BY REFERENCE phrases apply to the parameters that follow them until another BY CONTENT, BY VALUE or BY REFERENCE phrase is encountered. If none of these phrases appear before the first parameter, BY REFERENCE is assumed.

The BY VALUE phrase is not allowed for programs called with linkage type of program.

identifier-3
Must be defined as a level-01, level-77, or elementary data item in the File, Working-Storage, Local-Storage or Linkage Sections.

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)
- A procedure-pointer data item
- A DBCS data item
- A national data item
- A floating-point data item
- A date-time data item
- Reference modified, however, the length of the reference modified item must be known at compile time.

ADDRESS OF special register
For information about this special register, see page “ADDRESS OF Special Register” on page 126.

ADDRESS OF a data item
For information about this, see page “ADDRESS OF” on page 126.

LENGTH OF special register
The LENGTH OF special register contains the number of bytes used by a data item referenced by an identifier. For more information, see “LENGTH OF Special Register” on page 286.

literal-3
Can be:

- A nonnumeric literal
- A figurative constant
- A Boolean literal
- A DBCS literal
- A national literal.

float-literal-1
A floating-point literal is passed as an 8 byte internal float (COMP-2), unless the SIZE phrase is specified. For floating-point items the size phrase can be 4 or 8.

integer-1
Can be a signed or unsigned integer.

Integer-1 is passed as a binary value. If integer-2 is not specified then integer-1 will be passed as a 4-byte binary value. Integer-2 specifies the size of integer-1. This can be one of 1, 2, 4 or 8.

[End of IBM Extension]

[IBM Extension] LENGTH OF Special Register

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. However, the LENGTH OF special register cannot be applied to any operand that is reference modified.

The LENGTH OF operand may not be a function, but the LENGTH OF special register will be allowed in a function where an integer parameter is allowed.

If the LENGTH OF special register is used as the argument to the LENGTH function, the result will always be 4, independent of the argument specified for LENGTH OF.

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.

A date-time data item can be used in expressions using the LENGTH OF special register. The identifier may also be a type-name, or an item that is subordinate to a type-name.

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
```

Note: The number of bytes occupied by a COBOL item is also accessible through the intrinsic function LENGTH (see “LENGTH” on page 483). LENGTH supports nonnumeric literals in addition to data names.

[End of IBM Extension]

[IBM Extension] GIVING/RETURNING phrase

The GIVING/RETURNING phrase is not allowed for programs called with a linkage type of program. GIVING and RETURNING are equivalent.

identifier-4
The RETURNING data item which must be defined in the DATA DIVISION. The return value of the called program is implicitly stored into identifier-4. Identifier-4 cannot be reference modified.

Identifier-4 can be a date-time data item.

ADDRESS OF special register
For information about this special register, see page "ADDRESS OF Special Register" on page 126.

You can specify the RETURNING phrase on calls to ILE procedures that return a value. If you specify the RETURNING phrase on a CALL to a COBOL subprogram:

- The called subprogram must specify the RETURNING phrase on its PROCEDURE DIVISION header.
- Identifier-4 and the corresponding PROCEDURE DIVISION RETURNING identifier in the target program must have the same number of character positions and must be of the same USAGE and SIGN clause and category. If identifier-4 is defined using a TYPE clause, the item referenced in the GIVING/RETURNING phrase of the PROCEDURE DIVISION header of the called program must also be defined using a TYPE clause: the same type-name must be referenced in both TYPE clauses. When control returns to the calling program, identifier-4 or its ADDRESS of special register will contain the return value.

If an EXCEPTION or OVERFLOW occurs, identifier-4 is not changed.

The existence of the RETURNING phrase has no effect on the setting of the RETURN-CODE special register.

Items referenced in the RETURNING/GIVING phrase of the CALL statement cannot contain the TYPE phrase.

[End of IBM Extension]

ON EXCEPTION Phrase

This phrase handles the exceptions that result from program existence, program activation, authority, and storage if the original receiver of the exception is the caller. 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.

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 phrase 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 243.

CALL Statement Considerations

Call identifier

You can use CALL *identifier* (where *identifier* is not a procedure-pointer) to call a nested ILE COBOL program or a program object. The contents of the identifier determine, at run time, whether a nested program is called or a program object is called.

An open pointer that associates an identifier with an object is set the first time you use the identifier in a CALL statement.

If you carry out a call by an identifier to a program object that you subsequently delete or rename, you must use the CANCEL statement to null the open pointer associated with the identifier. This ensures that when you next use the identifier to call your program object, the associated open pointer will be set again.

The value of the open pointer changes if you change the value of the identifier and perform a call using this new value.

CALL procedure-pointer

You can perform a static procedure call or a dynamic program call using the CALL *procedure-pointer* statement.

Before using the CALL *procedure-pointer* statement, you must use the Format 6 SET Statement to set the value of the *procedure-pointer* data item. To set the *procedure-pointer* data item to an ILE procedure, specify LINKAGE TYPE IS PROCEDURE in the SET statement. To set the *procedure-pointer* data item to a program object, specify LINKAGE TYPE IS PROGRAM.

You can also use the LINKAGE TYPE clause of the SPECIAL-NAMES paragraph or the LINKLIT parameter of the CRTCBLMOD or CRTBNDCBL command to determine which type of object the *procedure-pointer* data item is set to. Refer to “LINKAGE TYPE Clause” on page 92 for information on using the LINKAGE TYPE clause or the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*for information on using LINKLIT parameter.

Length of Parameters

If the length of any parameter (in bytes), as defined in the calling program, does not match the length expected by the called program, unexpected results could occur in the called or calling program. See the section on "Passing and Sharing Data between Programs" in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*for details.

Program Termination Statements

The STOP RUN, EXIT PROGRAM, and GOBACK statements are used to return control from a called ILE COBOL program. The action taken for each program termination statement when an error occurs, or a program ends depends on whether control is returned from a main program or a subprogram. For details on the behavior of the EXIT PROGRAM, STOP RUN, and GOBACK statements under various conditions, see the section "Returning from an ILE COBOL Program" in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* . For details on each individual program termination statement, see:

- “EXIT PROGRAM Statement” on page 319

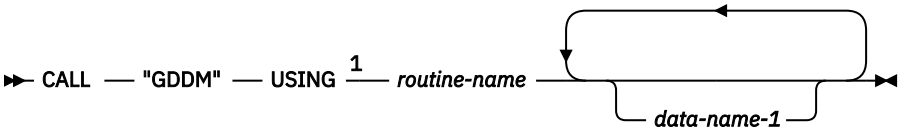
- “GOBACK Statement” on page 320
- “STOP Statement” on page 414

[IBM Extension] IBM i Graphics Support

You can use the CALL statement to access the following IBM i 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 Statement - Format

Notes:

[1] IBM Extension

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.

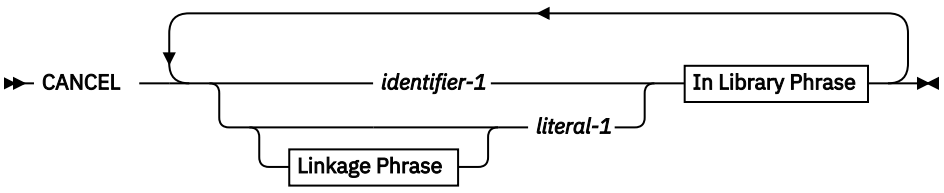
```
MOVE "FSINIT" TO OS-400-GRAPHICS-ROUTINE-NAME.
CALL "GDDM" USING OS-400-GRAPHICS-ROUTINE-NAME.
   .
   .
MOVE "GSFLD" TO OS-400-GRAPHICS-ROUTINE-NAME.
CALL "GDDM" USING OS-400-GRAPHICS-ROUTINE-NAME,
                          PIC-ROW, PIC-COL,
                          PIC-DEPTH, PIC-WIDTH.
```

For more information about graphics routines and their parameters, see the *GDDM Programming Guide* book and the *GDDM Reference*.

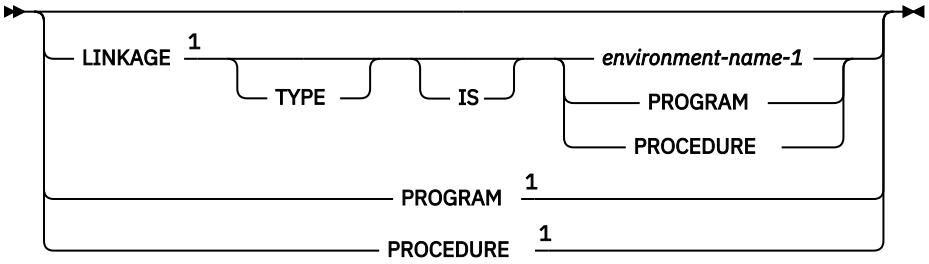
[End of IBM Extension]

CANCEL Statement

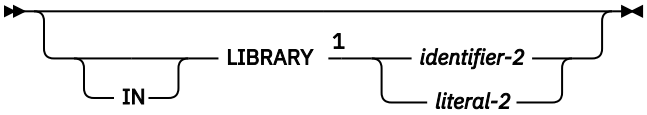
The CANCEL statement ensures that the next time the referenced subprogram is called it will be entered in its initial state.



Linkage Phrase



In Library Phrase



CANCEL Statement - Format

Notes:

[1] IBM Extension

literal-1

The name of the subprogram to be canceled. Literal-1 can contain an extended name. The CANCEL linkage determines the type of program to be canceled and thus also restricts the content and size of literal-1. The linkage made is either to a program object or an ILE procedure. If the linkage is to a program object, then literal-1 must be nonnumeric, uppercase (except for an extended system name), 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.

If the linkage is to an ILE procedure, then literal-1 must be nonnumeric and at most 250 characters long. Depending on the compiler option *MONOPRC, literal-1 might need to be uppercase and conform to the rules for formation of program-names. The literal must specify the program-name of the called subprogram.

identifier-1

Must be an alphanumeric data item where the following rules apply:

- If the linkage is to a program object, the contents of identifier-1 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 the linkage is to a procedure, the first 250 characters of identifier-1 are used.
- If the compiler option *MONOPRC is specified, the contents of identifier-1 need to be uppercase and must conform to the rules for formation of program-names.

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.

[IBM Extension] IN LIBRARY Phrase

This phrase is only valid for canceling an IBM i program object. That is, a linkage of type program must be specified, either implicitly or explicitly, on the CANCEL statement.

identifier-2

Must be an alphanumeric data item. The contents of identifier-2 must represent a valid IBM i library name. IBM i library names are at most 10 characters long. The first 10 characters of identifier-2 are used to form the library name.

literal-2

Must be nonnumeric and can be a maximum of 10 characters long.

Identifier-2 and literal-2 are *not* affected by the *MONOPRC compiler option, and can contain an IBM i extended name.

[End of IBM Extension]

LINKAGE TYPE Phrase

[IBM Extension]

The LINKAGE TYPE phrase is used to specify the type of program that the CANCEL statement targets. It could target a program object (*PGM) or an ILE procedure.

environment-name-1
The type of program that the CANCEL statement will affect. Environment-name-1 can be defined as:

PGM
A program object (*PGM)

PRC
An ILE procedure

PROGRAM
A program object (*PGM) is canceled.

PROCEDURE
An ILE procedure is canceled.

If the LINKAGE TYPE phrase is not specified on the CANCEL statement, the type of program canceled can be changed by specifying one of: the LINKAGE TYPE clause of the SPECIAL-NAMES paragraph, or the LINKLIT parameter of the CRTCBLMOD or CRTBNDCBL command.

[End of IBM Extension]

After a CANCEL statement for a called subprogram has been executed, that subprogram no longer has a logical connection to the program. The contents of data items in external data records described by the subprogram are not changed when a subprogram is canceled. 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 INTERNAL files.

You can cancel a called subprogram in any of the following ways:

- By referencing it as the operand of a CANCEL statement
- By terminating the run unit of which the subprogram is a member (This can be done by a STOP RUN in the same run unit or by a GOBACK from the main program of the run unit.)
- By executing an EXIT PROGRAM statement in the called subprogram if that subprogram possesses the INITIAL attribute
- By executing the GOBACK statement in the called subprogram if that subprogram possesses the INITIAL attribute.

A CANCEL statement operates only on the program specified, and not on any program that may have been called by the canceled program.

Called subprograms may contain CANCEL statements. A called subprogram must not contain a CANCEL statement that directly or indirectly cancels its calling program or any other program higher than itself in the calling hierarchy. If a called subprogram attempts to cancel its calling program, the CANCEL statement in the subprogram is ignored.

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. 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.)
```

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 both cases, control passes to the next statement.

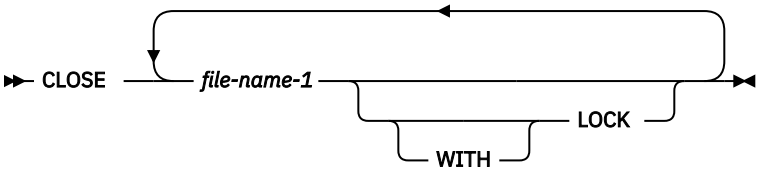
See the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information on canceling procedures and program objects.

CLOSE Statement

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

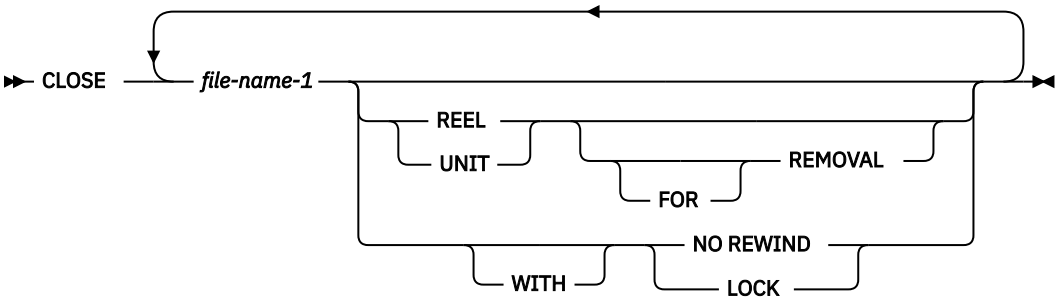
CLOSE Statement - Format 1

CLOSE Statement - Format 1



CLOSE Statement - Format 2 - Tape Files

CLOSE Statement - Format 2 - Tape Files



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.

CLOSE Statement Considerations

A CLOSE statement may be executed only for a file in an open mode. After successful execution of a CLOSE statement without the REEL or UNIT phrase:

- The record area associated with the file-name-1 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.

The following considerations apply to the use of the CLOSE statement:

- 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 without a REEL or UNIT phrase has not been processed before the end of the run unit is reached, or before the program is cancelled, then the file is closed implicitly.
- 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.
- 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 250.
- **For Relative Files Only**: 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.

WITH LOCK Phrase

COBOL ensures that this file cannot be reopened by this COBOL program during this processing of the program. External files closed WITH LOCK cannot be opened again within the run unit. This includes any other programs that have defined the external file.

Special Considerations for Device Type TAPEFILE Only

Files with device type TAPEFILE can be divided into the following two categories:

Sequential Single Volume

A sequential file that is entirely contained on one volume (reel). More than one file may be present on this volume.

Sequential Multivolume

A sequential file that is contained on more than one volume. The file either may contain more data than can be held on a single volume, or it may have been deliberately divided over multiple volumes.

The following phrases apply only to device type TAPEFILE:

- NO REWIND phrase
- REEL or UNIT phrase
- FOR REMOVAL phrase

If none of these phrases is specified, the CLOSE statement causes the current volume to be positioned at its beginning.

For sequential multivolume files, a CLOSE statement that does not include a REEL or UNIT phrase has no effect on any volume other than the current volume.

NO REWIND Phrase

The current volume is left in its present position.

REEL or UNIT Phrase

When the REEL or UNIT phrase is specified for an output file, it indicates that a sequential multivolume file is being created, and that no more records are to be written to the current volume of the file. The following processing takes place:

1. Standard labels are written at the end of the current volume.
2. A message is issued asking for a new volume to be mounted to receive the continuation of the file.
3. Standard labels are written at the start of the new volume.
4. The next WRITE statement that is processed writes a record to the newly mounted volume.

When the REEL or UNIT phrase is specified for a sequential multivolume file that is open for input, the current volume is positioned to read the standard labels. If this is the last volume of the file, the program continues, and the next READ statement that is processed will cause the AT END condition to occur. If this is not the last volume of the file:

1. A message is issued asking for the next volume of the file to be mounted.
2. The standard labels at the start of the next volume are processed.
3. The next READ statement that is processed requests the first record on the newly mounted volume.

The REEL or UNIT phrase is optional for sequential single volume files open for input. It is syntax-checked only, and performs no function at run time.

FOR REMOVAL Phrase

For sequential multivolume files, the addition of the FOR REMOVAL phrase to the REEL or UNIT phrase causes the current volume to be rewound and unloaded. The system is then notified that the volume is logically removed from this run unit. The volume can be addressed again, however, after the file has been closed by a CLOSE statement without the REEL or UNIT phrase, and then reopened.

The use of the FOR REMOVAL phrase is optional for sequential single volume files open for input. It is syntax-checked only, and performs no function at run time.

[IBM Extension]

For sequential multivolume files, the system will always rewind and unload the volume when the REEL or UNIT phrase is specified on the CLOSE statement, even if the FOR REMOVAL phrase is not included.

A file will be closed implicitly if the end of the run unit is reached, or if the program is cancelled, before a CLOSE statement without a REEL or UNIT phrase has been processed. If this occurs, then the current volume will be left positioned as defined by the ENDOPT keyword held in the system description of the file. This keyword, which may take the values LEAVE, REWIND, or UNLOAD, is set up when the file description is created by the CRTTAPF command. It may be changed using the CHGTAPF command, or overridden using the OVRTAPF command.

[End of IBM Extension]

[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 - Format

When the COMMIT statement is executed, all changes made to files under commitment control, for the current commitment definition 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, 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 current commitment definition since the last commitment boundary for files under commitment control are released and the records become available. Commitment control can be scoped at the job level or the activation group level. Commitment control scopes to the activation group by default. This is important when your application involves non-ILE COBOL programs that run in a different activation group, such as a CL program.

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 *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

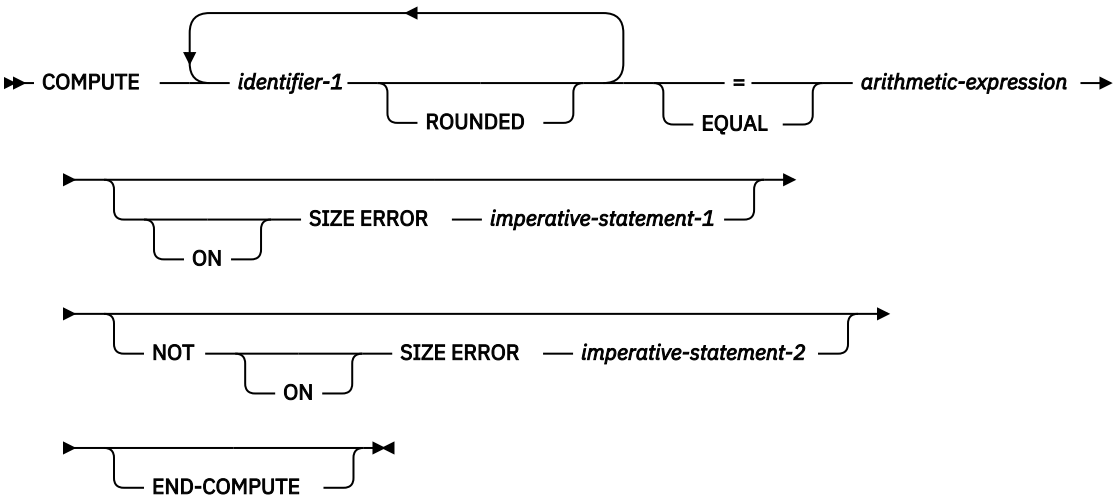
[End of IBM Extension]

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.

COMPUTE Statement - Format



If portability is desired, however, you should use ADD, SUBTRACT, MULTIPLY, and DIVIDE rather than COMPUTE. This is because of potentially different system-specific intermediate results.

When arithmetic operations are combined, the COMPUTE statement may be more efficient than the separate arithmetic statements written in series.

identifier-1

Must be either elementary numeric item(s) or elementary numeric-edited item(s).

[IBM Extension] Can be an elementary floating-point data item. [End of IBM Extension]

arithmetic-expression

Can be any arithmetic expression, as defined in “Arithmetic Expressions” on page 223.

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, numeric function, or numeric 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

See “ROUNDED Phrase” on page 246.

SIZE ERROR Phrases

See “SIZE ERROR Phrases” on page 246.

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 243.

CONTINUE Statement

The CONTINUE statement allows you to specify a no operation statement. CONTINUE indicates that no executable instruction is present.

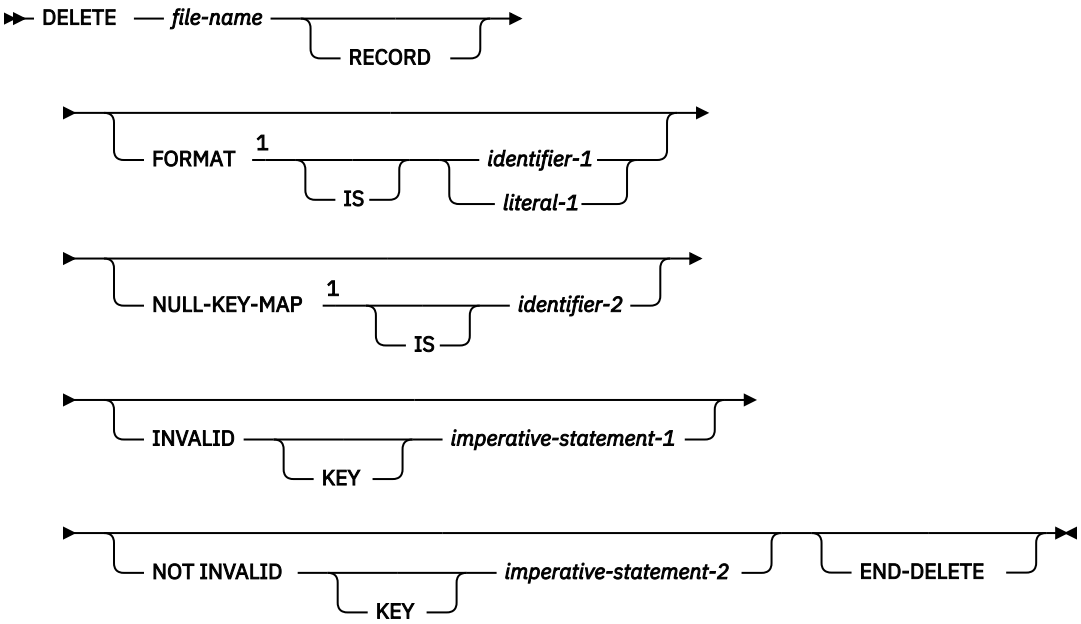
CONTINUE Statement - Format

The CONTINUE statement can be used anywhere a conditional statement or an imperative statement can 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.



DELETE Statement - Format

Notes:

[1] IBM Extension

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 (or the content of the data item referenced by the data-name specified in the DEPENDING ON phrase of the RECORD clause associated with *file-name*).

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.

[IBM Extension] DELETE Statement Considerations

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. For more information on the OVRDBF command, see the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

[End of IBM Extension]

Sequential Access Mode

For an indexed or relative file in sequential access mode,

- When the DELETE statement is processed, the system logically removes the record retrieved and locked by the READ statement.

  The last input/output statement must have been a successfully processed READ statement **without** the NO LOCK phrase.

  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.

  [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. [End of IBM Extension]

  See the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for information about file and record locking.
- 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 250.

Random or Dynamic Access Mode

In random and dynamic access modes, the results of using the DELETE statement depend on the file organization.

When it is a relative file, the system logically removes the record identified by the contents of the RELATIVE KEY data item. The space is then available for a new record with the same RELATIVE KEY value. If the file does not contain such a record, an INVALID KEY condition exists.

On an indexed file, 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 fields in the [2] for the format specified by the FORMAT phrase are 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. [End of IBM Extension]

[IBM Extension] 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.

[2] The key fields in the record area are the locations in the buffer selected in accordance with a record format or specification in order to build a search argument.

- 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.

If the value of the RECORD KEY data item has been changed since the record was read:

- The file status key, if defined, is set to 21.
- An INVALID KEY condition exists.
- The DELETE statement is not processed.

[End of IBM Extension]

[IBM Extension] FORMAT Phrase

The FORMAT phrase applies only to indexed files of device type DATABASE. It is required when processing a file that has multiple record formats and has unique keys. If the record key is defined with duplicates, the FORMAT phrase is incorrect and is ignored.

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 a file, a FILE STATUS of 9K is returned and a USE procedure is invoked, if applicable for the file.

[End of IBM Extension]

[IBM Extension] NULL-KEY-MAP IS Phrase

For a description of the NULL-KEY-MAP IS phrase, refer to the description given for the START statement, “NULL-KEY-MAP IS Phrase” on page 410.

[End of IBM Extension]

INVALID KEY Phrase

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 250.

NOT INVALID KEY Phrase

After the successful processing of a DELETE statement for which there is a NOT INVALID KEY phrase, control transfers to the imperative statement associated with the phrase.

END-DELETE Phrase

This explicit scope terminator delimits the scope of the DELETE statement. It permits a conditional DELETE statement to be nested in another conditional statement. END-DELETE can also be used with an imperative DELETE statement.

DISPLAY Statement

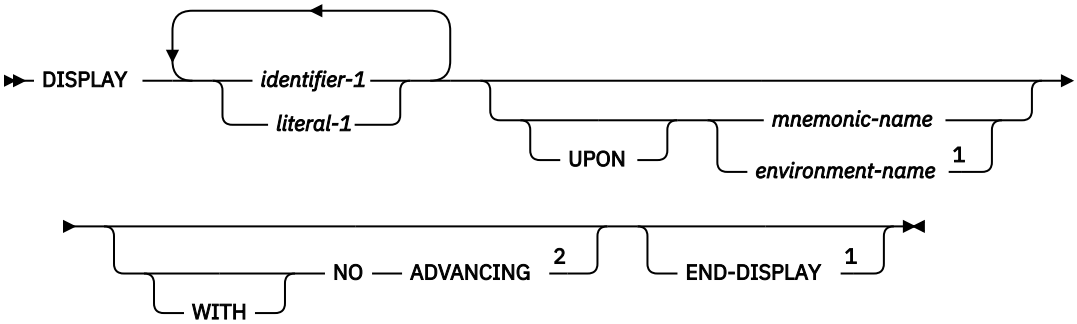
The DISPLAY statement transfers the contents of each operand to the output device. The contents are displayed on the output device in the order, left to right, in which the operands are listed.

[IBM Extension]

- Format 2 - Local Data Area
- Format 3 - Workstation I/O
- Format 4 - Session I/O
- Format 5 - Data Area

[End of IBM Extension]

Format 1 - Data Transfer



DISPLAY Statement - Format 1 - Data Transfer

Notes:

[1] IBM Extension

[2] Syntax-checked only.

identifier-1

[IBM Extension] If the description of identifier-1 contains a TYPE clause, the type-name specified in that clause must be elementary. [End of IBM Extension]

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, binary, floating-point, or date-time data (COMP, COMP-1, COMP-2, COMP-3, PACKED-DECIMAL, BINARY, COMP-4, or COMP-5) should not be displayed on a display station. Such data can contain display station control characters which can cause undesirable and unpredictable results.

- [IBM Extension] Can be an internal or external floating-point data item. Internal floating-point numbers are converted to external floating-point numbers for display, such that:
  - A COMP-1 item will display as if it had an external floating-point PICTURE clause of -.9(8)E-99
  - A COMP-2 item will display as if it had an external floating-point PICTURE clause of -.9(17)E-999

  It is possible that when an external floating-point literal is displayed, slight inaccuracies can result. This is especially true if the DISPLAY takes place after a MOVE. The floating-point data type is an

approximation, and when an external floating-point literal is moved, it is first converted to a true floating-point value (IEEE), which can also affect its accuracy.

For example, consider the following DISPLAY:

```
77 external-float-1 PIC +9(3).9(13)E+9(3).
    MOVE +123455779012.3453E+297 to external-float-1.
    DISPLAY "EXTERNAL-FLOAT-1=" external-float-1.
```

The displayed result after the MOVE is:

```
EXTERNAL-FLOAT-1=+123.4557790123452E+306
```

[End of IBM Extension]

- No other identifiers require conversion.

[IBM Extension]

- Elementary DBCS and national data items are transferred to the output device. DBCS, national, and SBCS operands can be specified using a single DISPLAY verb. Data output will be converted to the code set specified by the job's current CCSID.

[End of IBM Extension]

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] Floating-point literals are allowed. [End of IBM Extension]

[IBM Extension] Signed noninteger numeric literals are allowed. [End of IBM Extension]

[IBM Extension] DBCS and national literals are allowed. The ALL figurative constant can be used with DBCS and national literals in a DISPLAY verb. [End of IBM Extension]

UPON
The UPON phrase specifies a mnemonic-name that must be associated with either the workstation (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.

WITH NO ADVANCING
This phrase is syntax checked only and ignored. For a description of a functional WITH NO ADVANCING phrase see “Format 4 – Session I/O” on page 307.

DISPLAY Statement Behavior

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.
- [IBM Extension] If a DBCS or national operand must be split across multiple records, it splits only on a double-byte boundary. [End of IBM Extension]

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
Workstation	58 characters
System operator's message queue	58 characters

[IBM Extension] If a DBCS or national item or literal is specified in a DISPLAY verb, the size of the sending field is the total character count of all operands listed, with each DBCS or national character counted twice, plus all necessary shift codes for DBCS. [End of IBM Extension]

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 and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/

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.

```
                          Display Program Messages

JOB 000745/QPGMR/WS1 started on 02/17/92 at 14:50:22 in subsystem QINTER 1
SAMPLE PROGRAM MESSAGE FROM PREVIOUS CALL OF PROGRAM. 2
SAMPLE PROGRAM MESSAGE FROM CURRENT CALL OF PROGRAM. 2
```

- 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 either:

  – 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 *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* . For additional information on the RSTDSP parameter, see the CHGDSPF and CRTDSPF commands in the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

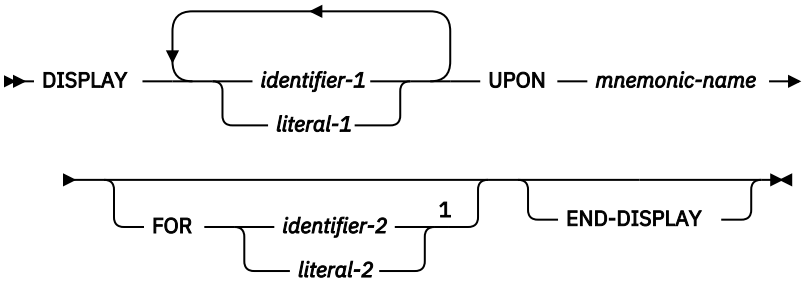
When a program started by a workstation operator sends a DISPLAY to the system operator's message queue (separate from the workstation), program processing is not suspended.

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	Workstation	Workstation

[IBM Extension] Format 2 – Local Data Area

This format is used to transfer data to the system-defined local data area created for a job.



DISPLAY Statement - Format 2 - Local Data Area

Notes:

[1] Syntax-checked only.

This format is only applicable when the mnemonic-name in the SPECIAL-NAMES paragraph is associated with the environment-name LOCAL-DATA.

The conversion and display rules for identifier-1 and literal-1 are described under "Format 1 - Data Transfer" on page 299. However, the restriction that if the description of identifier-1 contains a TYPE clause, that the type-name referenced be elementary, does not apply.

Identifier-2 and literal-2 cannot be floating-point data items.

Identifier-1 can be a date-time data item.

Identifier-1 can be a DBCS or national data item.

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 Programming* manual and the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

[End of IBM Extension]

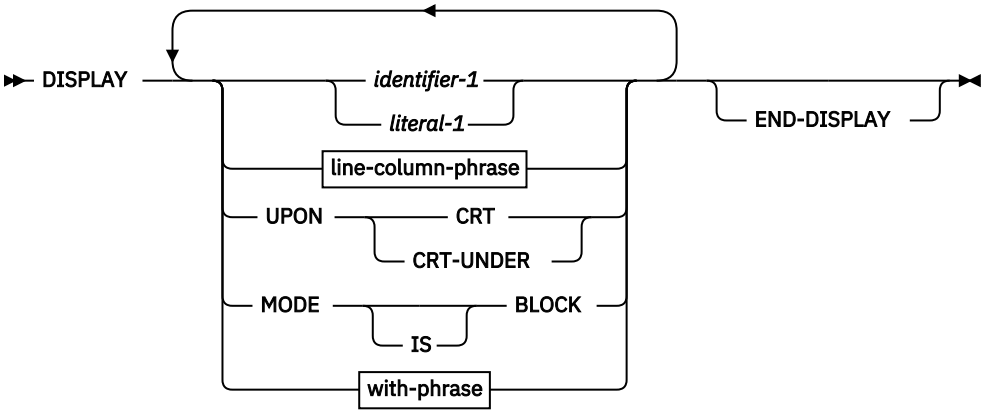
[IBM Extension] Format 3 – Extended DISPLAY Statement

A DISPLAY statement is considered an **extended** DISPLAY statement if it has one of the following:

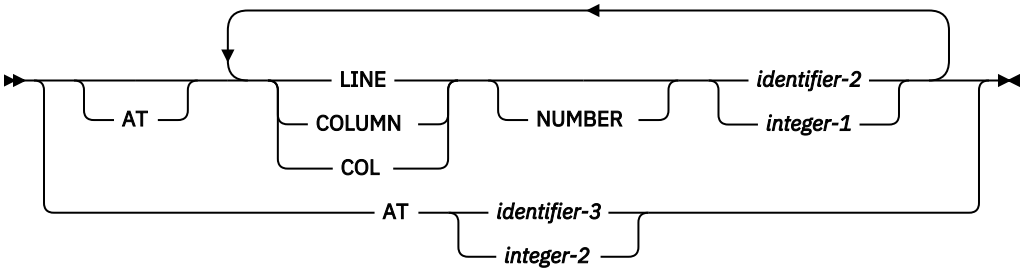
- An AT phrase
- An UPON CRT or UPON CRT-UNDER phrase
- A MODE IS BLOCK phrase
- A WITH phrase
- No UPON phrase and a CONSOLE IS CRT specified in the SPECIAL-NAMES paragraph.

A DISPLAY statement is considered a **standard** DISPLAY statement if it has one of the following:

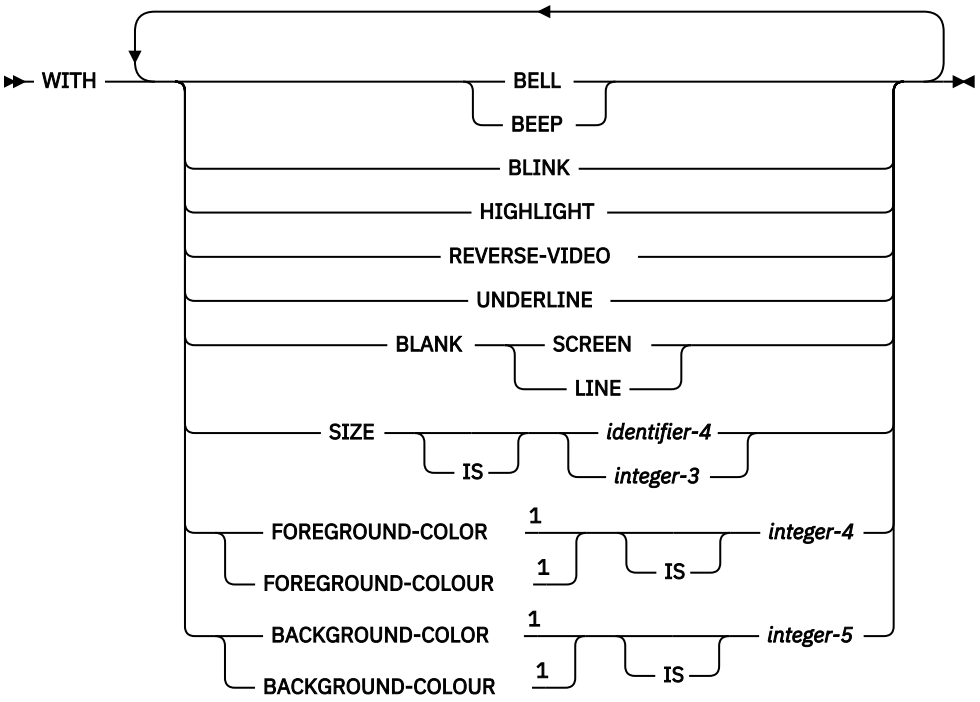
- An UPON phrase (other than UPON CRT or UPON CRT-UNDER)
- No UPON phrase and no CONSOLE IS CRT specified in the SPECIAL-NAMES paragraph.



line-column-phrase



with-phrase



DISPLAY Statement - Format 3 - Workstation I/O

Notes:

[1] Syntax-checked only.

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.

Identifier-1 cannot be a date-time item.

When identifier-1 does not fit within the screen, then alphanumeric data is truncated and numeric data is not put on the screen.

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.

[End of IBM Extension]

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.

COL is an abbreviation for COLUMN.

The LINE and COLUMN phrases can appear in any order.

identifier-2, integer-1
Identifier-2 and integer-1 must be unsigned numeric integers greater than or equal to zero, and less than 9 digits. If LINE or COLUMN is negative, the absolute value is taken.

Identifier-2 cannot be a floating-point data item.

Line and Column Combinations

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.

Identifier-3 cannot be a floating-point data item.

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.

Identifier-4 cannot be a floating-point data item.

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.

When the size specified for a numeric or a numeric-edited data item is less than that implied by the PICTURE clause, truncation of the rightmost positions occurs when the value is displayed, or predisplayed in the ACCEPT operation. The data item is then updated following the rules for the MOVE operation.

If you specify a SIZE literal whose value causes the field length to exceed the screen size, alphanumeric data will be truncated and numeric data will be ignored and not displayed.

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.

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.

SIZE Phrase Example

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.

Format 3 Considerations

A data item can contain a table whether or not MODE IS BLOCK has been specified. Fixed-length and variable-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.

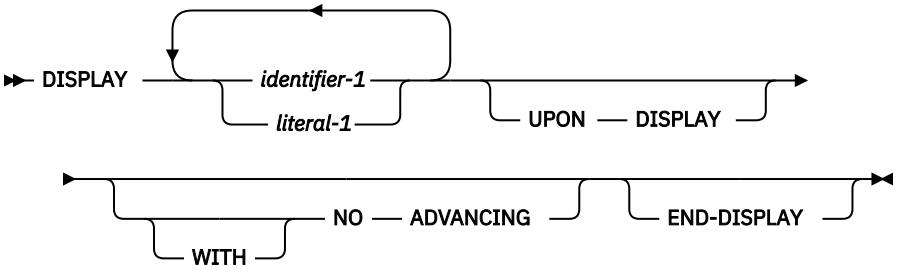
Some extended DISPLAY statement considerations also apply to the extended ACCEPT statement. (See “Extended ACCEPT and Extended DISPLAY Considerations” on page 271 for more information.)

The ILE COBOL 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 588.

[IBM Extension] Format 4 – Session I/O

This format is used to transfer data to the ILE common session manager.

DISPLAY Statement - Format 4 - Session I/O



This format is only applicable when the UPON DISPLAY phrase is specified or the CONSOLE IS DISPLAY clause is specified in the SPECIAL-NAMES paragraph.

The DISPLAY statement's literal operands or the contents of the DISPLAY statement's identifier operands, are written to the ILE common session manager. The data is written to the session manager according to the rules outlined in format 1 — Data Transfer (refer to the description of identifier-1 and literal-1 under “Format 1 - Data Transfer” on page 299). If the contents of identifier-1 or literal-1 span more than one line, writing of data continues at the first position of the next line of the ILE common session manager.

If the WITH NO ADVANCING phrase is not specified a new line character is written to the session manager; if it is specified, the session manager will be positioned immediately following the last character of the last operand displayed.

Identifier-1 can be date-time data item.

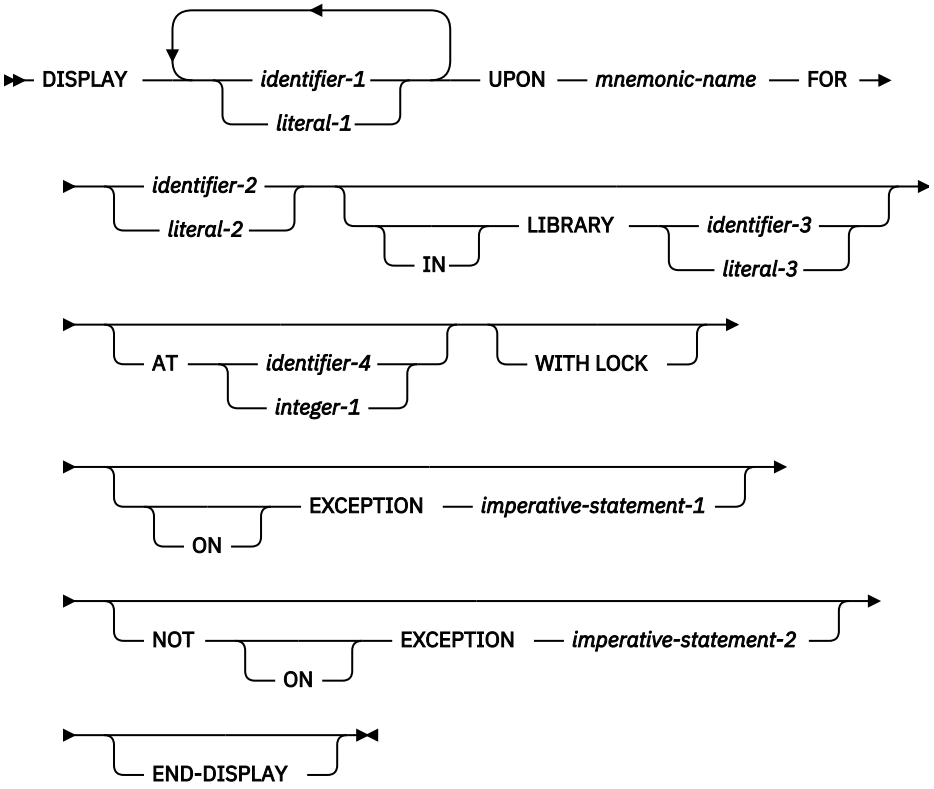
Identifier-1 can be a DBCS or national data item. If identifier-1 is national item, the output data will be converted to the code set specified by the job's current CCSID.

[End of IBM Extension]

[IBM Extension] Format 5 – Data Area

This statement is used to transfer data to the data area specified in the FOR phrase.

DISPLAY Statement - Format 5 - Data Area



This format is only applicable when the mnemonic-name in the SPECIAL-NAMES paragraph is associated with the environment-name DATA-AREA.

The DISPLAY statement's literal operands, or the contents of the DISPLAY statement's identifier operands, are written to the 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.

For more information and an example of how to use data areas, see the "Using Data Areas You Create" information in the "Passing Data Using Data Areas" section of the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

identifier-1/literal-1

The conversion and display rules for identifier-1 and literal-1 are described under “Format 1 - Data Transfer” on page 299.

Identifier-1 can be date-time data item.

[End of IBM Extension]

UPON

mnemonic-name in the SPECIAL-NAMES paragraph must be associated with the environment-name DATA-AREA.

When the UPON phrase is omitted, the DISPLAY statement sends output to the REQUESTOR.

FOR Phrase

Identifies the operating system data area to which to write information. If the data area specified cannot be located or accessed at run time an ON EXCEPTION condition exists.

identifier-2
Must be an alphanumeric data item. The contents of identifier-2 must represent a valid operating system data area name. Operating system data area names are at most 10 characters long, thus the first 10 characters of identifier-2 are used to form the data area name.

literal-2
Must be nonnumeric and at most 10 characters long.

IN LIBRARY Phrase

Is used to specify the name of the operating system library in which the data area is to be found. The special values *LIBL (search using the job's library list) or *CURLIB (search the current library) may be specified. If the LIBRARY phrase is omitted, the job's library list is used to search for the data area.

identifier-3
Must be an alphanumeric data item. Since operating system library names are at most ten characters long , only the first ten characters of identifier-3 are used to form the library name.

literal-3
Must be nonnumeric and at most 10 characters long.

Identifier-2, identifier-3, literal-1, and literal-2 are *not* affected by the *MONOPRC compiler option. They can contain an operating system quoted name (for details, see "Rules for Specifying Names" in the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/).

AT Phrase

The AT phrase indicates the starting position in the data area to which text is written.

If the AT phrase is not specified, a starting position of 1 is assumed.

identifier-4, integer-1
Identifier-4 and integer-1 must be positive numeric integers with a value that ranges from 1 to the maximum data area size (2000).

WITH LOCK Phrase

Before data is transferred to the specified data area in the FOR phrase, a lock must be obtained. If a lock cannot be obtained, the data is not transferred, and an ON EXCEPTION condition exists.

To maintain a lock on the data area after the transfer of data, specify this phrase. If a lock existed on the data area prior to this statement and the statement did not contain a WITH LOCK phrase, the lock is released.

(NOT) ON EXCEPTION

If an error occurs while accessing the data-area, 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 data area is accessed successfully, any imperative statement specified in the NOT ON EXCEPTION phrase is processed.

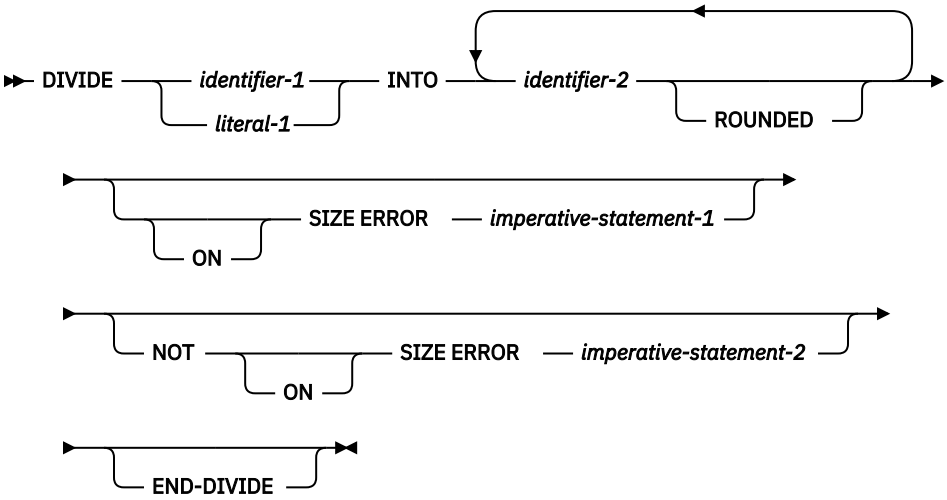
END-DISPLAY Phrase

The END-DISPLAY explicit scope terminator serves to delimit the scope of the DISPLAY statement. END-DISPLAY permits a conditional DISPLAY statement to be nested in another conditional statement. END-DISPLAY can also be used with an imperative DISPLAY statement. For more information, see "Delimited Scope Statements" on page 243.

DIVIDE Statement

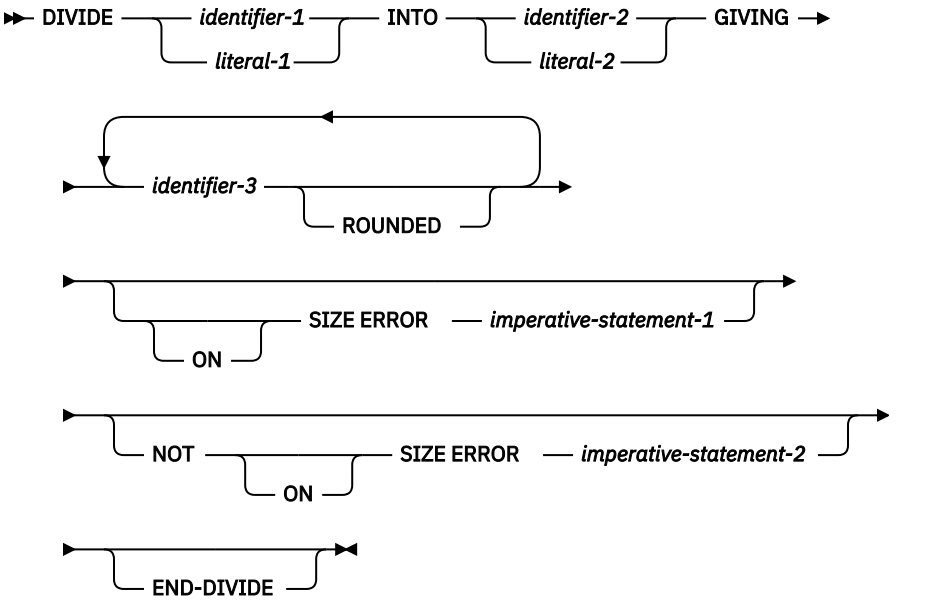
The DIVIDE statement divides one numeric data item into or by one or more others, and stores the result in the quotient and remainder.

DIVIDE Statement - Format 1 - INTO



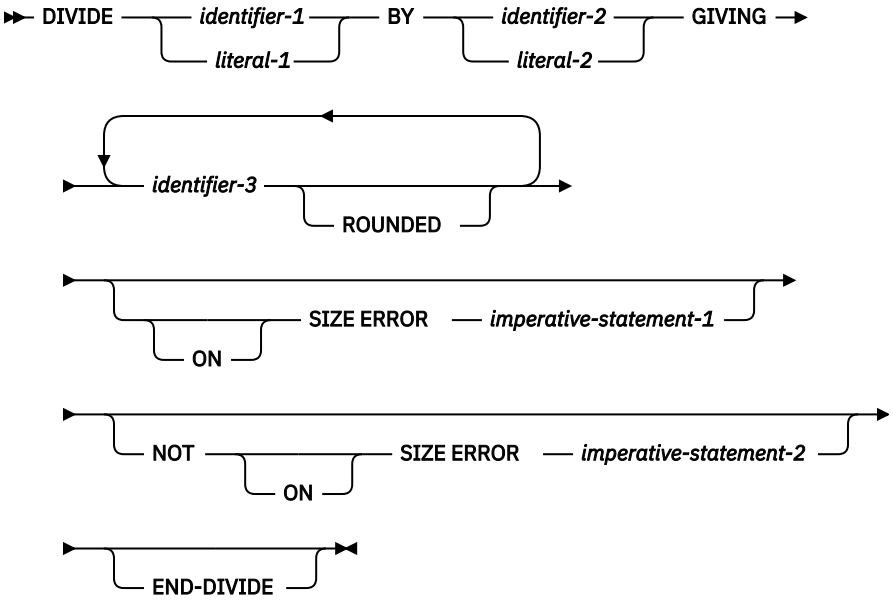
In Format 1, the value of identifier-1 or literal-1 is divided into the value of identifier-2; the quotient is then placed in identifier-2. This process is repeated for each successive occurrence of identifier-2.

DIVIDE Statement - Format 2 - INTO GIVING



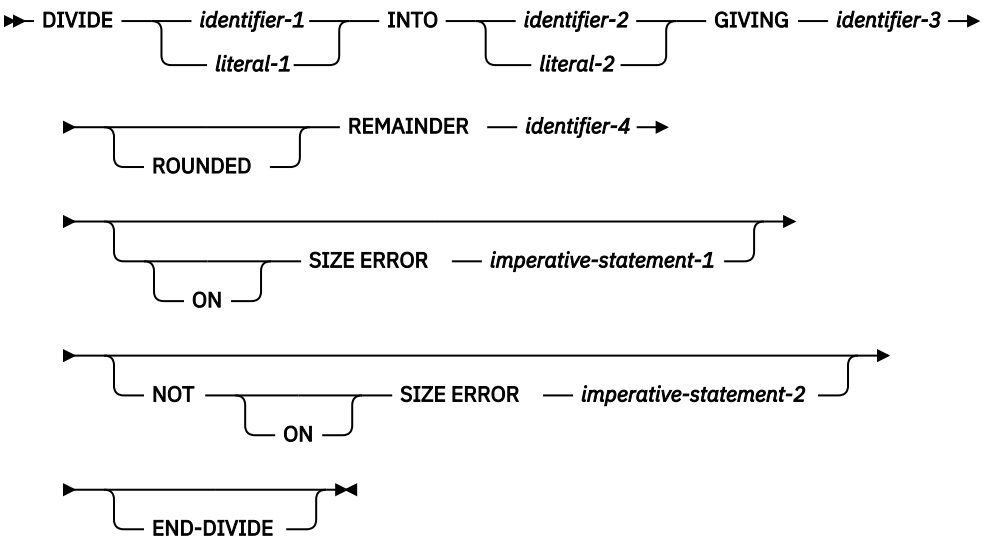
In Format 2, the value of identifier-1 or literal-1 is divided into the value of identifier-2 or literal-2. The value of the quotient is stored in each data item referenced by identifier-3.

DIVIDE Statement - Format 3 - BY GIVING



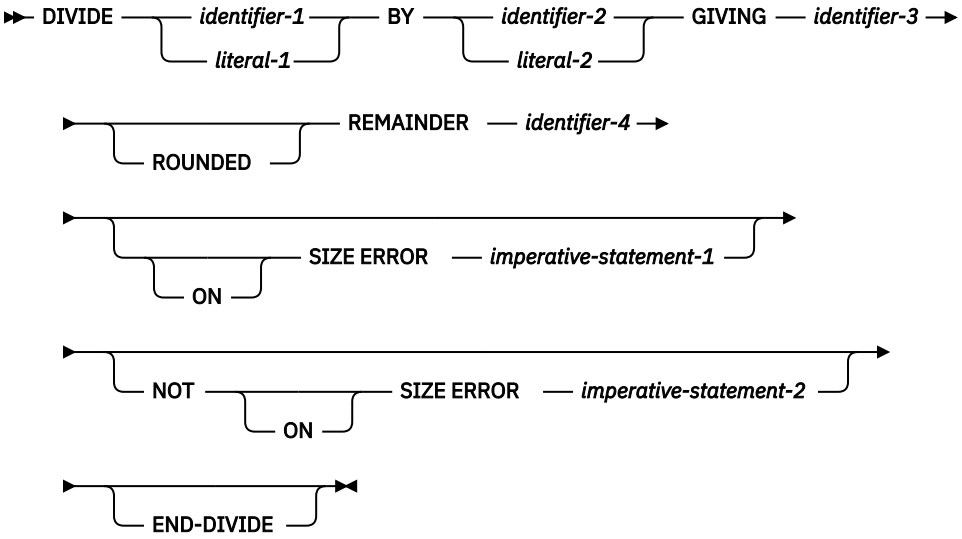
In Format 3, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2. This quotient is stored in each data item referenced by identifier-3.

DIVIDE Statement - Format 4 - INTO GIVING REMAINDER



In Format 4, the value of identifier-1 or literal-1 is divided into identifier-2 or literal-2. This quotient is stored in identifier-3, and the value of the remainder is stored in identifier-4.

DIVIDE Statement - Format 5 - BY GIVING REMAINDER



In Format 5, the value of identifier-1 or literal-1 is divided by identifier-2 or literal-2. This quotient is stored in identifier-3, and the value of the remainder is stored in identifier-4.

For all Formats:

identifier-1, identifier-2
Must be an elementary numeric item.

identifier-3, identifier-4
Must be an elementary numeric or numeric-edited item.

literal1, literal2
Must be a numeric literal.

The composite of operands is determined by superimposing all of the receiving data items, excluding the REMAINDER data item. For more information on the composite of operands, see the "Size of Operands" on page 247.

[IBM Extension]

In Formats 1 through 3, floating-point data items and literals can be used anywhere that a numeric data item or literal can be specified.

In Formats 4 and 5, floating-point data items or literals cannot be used.

[End of IBM Extension]

ROUNDED Phrase

For Formats 1, 2, and 3, see "ROUNDED Phrase" on page 246.

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.

[IBM Extension] The REMAINDER phrase is not valid if the receiver or any of the operands are floating-point items. [End of IBM Extension]

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

Divide by zero will trigger ON SIZE ERROR if ON SIZE ERROR is coded, otherwise the result of the division by zero is undefined.

For Formats 1, 2, and 3, see "SIZE ERROR Phrases" on page 246.

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 "NOT ON SIZE ERROR" on page 247.

END-DIVIDE Phrase

This explicit scope terminator delimits the scope of the DIVIDE statement. END-DIVIDE converts a conditional DIVIDE statement into an imperative statement so that it can be nested in another conditional statement.

For more information, see "Delimited Scope Statements" on page 243.

[IBM Extension] DROP Statement

The DROP statement releases a program device that has been acquired by a TRANSACTION file.

DROP Statement - Format

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.

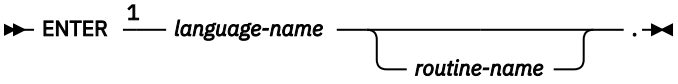
If the DROP statement is unsuccessful, any applicable USE AFTER EXCEPTION/ERROR procedures are executed.

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 *IBM Rational Development Studio for i: ILE COBOL Programmer'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. It is syntax checked only.



ENTER Statement - Format

Notes:

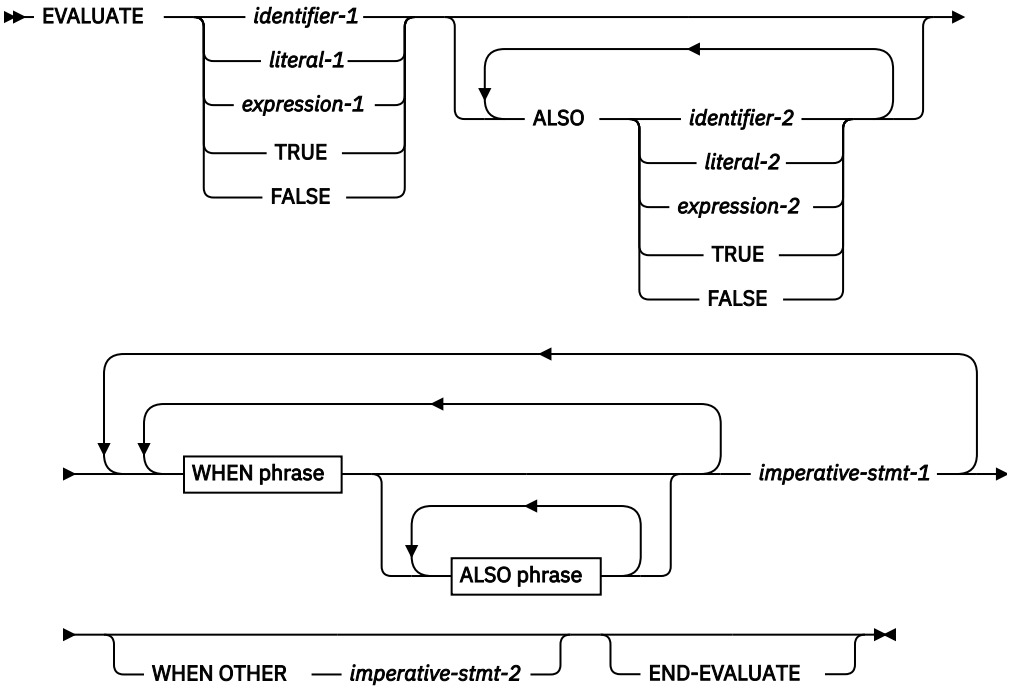
[1] Syntax-checked only.

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.

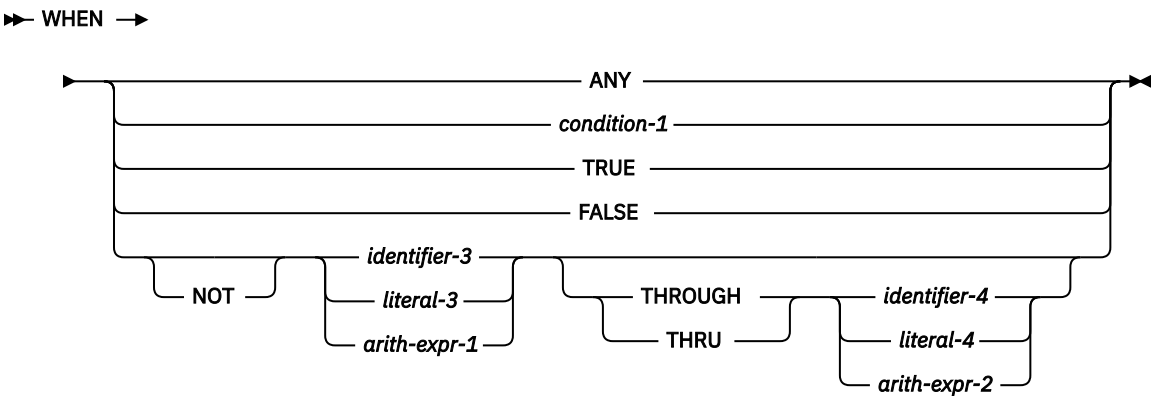
routine-name
Must follow the rules for formation of a user-defined word. At least one character must be alphabetic.

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.

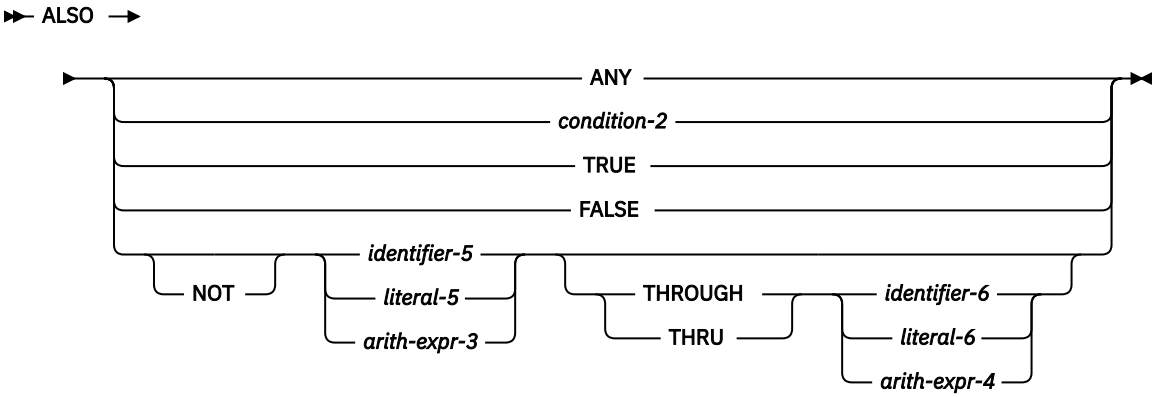


WHEN phrase



EVALUATE Statement - Format

ALSO phrase



The following example shows the coding for an EVALUATE statement and the equivalent coding for an IF statement.

Coding Examples

Simple 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.
```

The following is a more complex example of an EVALUATE statement and the equivalent IF statement.

Complex Example of the EVALUATE Statement:

```
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.
```

Interpreting Selection Subjects and Selection Objects

Operands before the WHEN phrase

Individually, they are called selection **subjects**.
Collectively, they are called a **set** of selection subjects.

Operands in the WHEN phrase

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.
- Conditional expressions may be simple or complex conditions.

[IBM Extension]

- Where numeric literals are permitted, floating-point literals are permitted.

- Identifiers can reference items whose usage is implicitly or explicitly defined as POINTER or PROCEDURE-POINTER.
- Identifiers can reference DBCS, national, or floating-point data items.
- Identifiers can reference date-time data items.
- Where nonnumeric literals are permitted, DBCS and national literals are permitted also.

[End of IBM Extension]

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 243.

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 223.)

  Note: Comparing one arithmetic expression to another is system-specific. The truth status of the comparison may depend on the intermediate results created on that system.
- 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 225.)
- 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.

  Note: Results of comparisons with nonnumeric operands may not be consistent across systems, if the comparisons depend on the system's native collating sequence.
- 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.

EXIT Statement - 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 to which control is transferred:

- When control reaches an EXIT paragraph that is the end of a range of procedures governed by an active PERFORM or USE statement, control is transferred in accordance with the rules for that PERFORM or USE statement.
- 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. It must not be used in the range of a global declarative unless it is in a different program called by the statement in the range of the global declarative.

EXIT PROGRAM Statement
Notes:

[1] IBM Extension

AND CONTINUE RUN UNIT
Exits the called program without stopping the run unit.

If control reaches an EXIT PROGRAM statement in a program that does not possess the INITIAL attribute while operating under the control of a calling program, control returns to the CALL statement of the calling program.

The program state of the calling program is identical to that which existed at the time it executed the CALL statement except that the contents of data items and the contents of the data files shared between the two programs 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.

The execution of an EXIT PROGRAM statement in a called program that possesses the INITIAL attribute performs an implicit CANCEL of the referenced program.

If control reaches an EXIT PROGRAM statement *without* the continue phrase in the main program, control passes through the exit point to the next executable statement.

The EXIT PROGRAM statement should appear as the last statement in a series of imperative statements within a sentence.

When there is no next executable statement in a called program, an implicit EXIT PROGRAM statement is assumed, and executed.

The RETURN-CODE special register can be used to pass return code information from a program to its caller. See “RETURN-CODE Special Register” on page 415 for further information.

[IBM Extension] AND CONTINUE RUN UNIT Phrase

If control reaches an EXIT PROGRAM statement *with* the continue phrase in the main program, control passes to the CALL statement of the calling program. In a named activation group:

- The activation group remains active
- The main program is left in its last used state, except that the ends of the ranges of all PERFORM statements executed by that called program are considered to have been reached.

However, in a *NEW activation group when a main program returns control to the caller, the activation group is ended. The activation group will close all files scoped to the activation group. Any pending commit operation scoped to the activation group will be implicitly committed. All resources allocated to the activation group will be returned back to the system. As a result of the activation group ending, all programs that were active in the activation group are placed in their initial state.

[End of IBM Extension]

[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 called program.

GOBACK Statement - Format

A GOBACK statement should appear as the only statement, or as the last of a series of imperative statements, in a sentence because statements following the GOBACK statement are not executed.

If control reaches a GOBACK statement while a CALL statement is active, control returns to the point in the calling program immediately following the CALL statement, as in the EXIT PROGRAM statement.

The RETURN-CODE special register can be used to pass return code information before executing a GOBACK statement. See "RETURN-CODE Special Register" on page 415.

In a multi-threaded environment (for example, when the THREAD(SERIALIZE) PROCESS option has been specified), the GOBACK statement returns to the caller of the program without terminating the thread and run unit. For further information, see the chapter on Preparing ILE COBOL Programs for Multithreading in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

For further information about COBOL run units, see the chapter on Calling and Sharing Data Between ILE COBOL Programs in the *IBM Rational Development Studio for i: ILE COBOL Programmer'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:

- Unconditional
- Conditional
- Altered

If procedure-name or procedure-name-1 are within a declarative procedure, neither can reference another declarative procedure or any nondeclarative procedure. 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 procedures associated with it.

Unconditional GO TO

An 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 277.)

GO TO Statement - Format 1 - Unconditional

procedure-name
Must be a section or paragraph in the same Procedure Division as the GO TO statement.

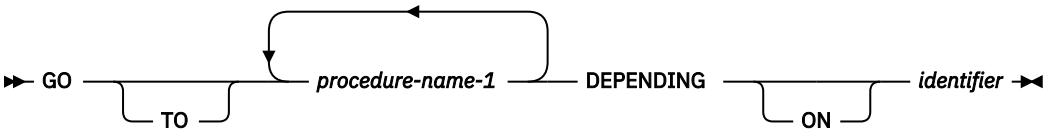
An unconditional GO TO statement, when it appears in a sequence of imperative statements, must be the last statement in the sequence.

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 data item referenced by the identifier.

GO TO Statement - Format 2 - Conditional



procedure-name-1
Must be a section or paragraph in the Procedure Division.

identifier
Must be a numeric elementary data item which is an integer.

[IBM Extension] Cannot be a floating-point data item. [End of IBM Extension]

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.

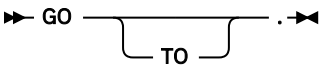
If the value of identifier is anything other than a value within the range of 1 through n (where n is the number of procedure-names specified in this GO TO statement), no control transfer occurs. Instead, control passes to the next statement in the normal sequence of execution.

Altered GO TO

The altered GO TO statement transfers control to the first statement of the paragraph named in the ALTER statement.

An ALTER statement referring to the paragraph containing this GO TO statement must have been executed before this GO TO statement is executed. You cannot specify the altered GO TO statement in a program that has the RECURSIVE attribute.

GO TO Statement - Format 3 - Altered

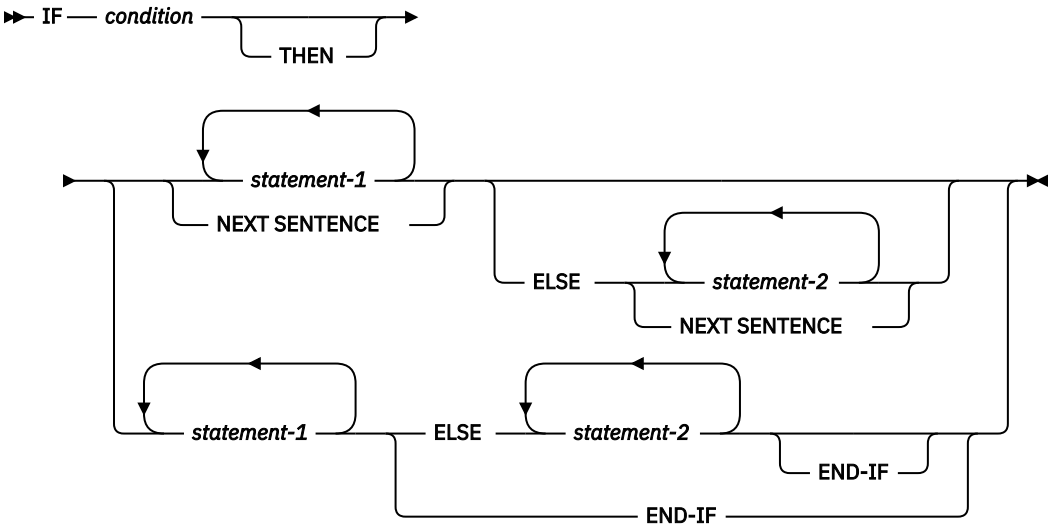


The altered GO TO statement can not be specified in a program that has the RECURSIVE attribute.

IF Statement

The IF statement evaluates a condition and provides for alternative actions in the object program, depending on the evaluation.

IF Statement — Format



The scope of an IF statement can be terminated by any of the following:

- An END-IF phrase at the same level of nesting
- A separator period
- If nested, by an ELSE phrase associated with an IF statement at a higher level of nesting

condition
May be any simple or complex condition, as described in "Conditional Expressions" on page 225.

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 243.

Transferring Control

If the 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 (implicit or explicit) 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.

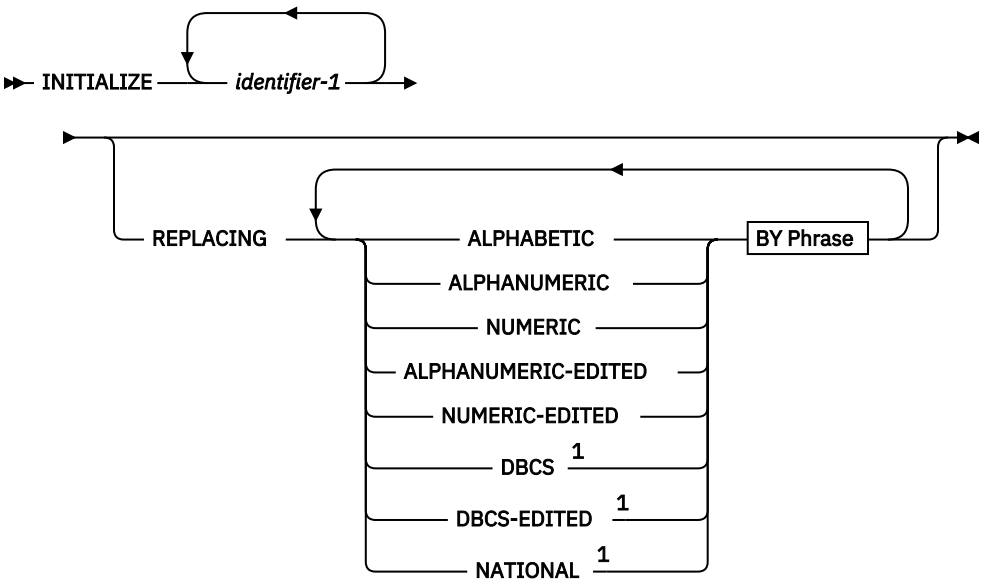
Nested IF Statements

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.

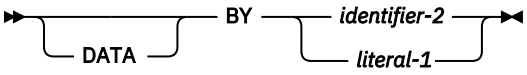
IF statements contained within IF statements are considered as paired IF, ELSE, and END-IF combinations, proceeding from left to right. Thus, any ELSE or END-IF encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE or END-IF.

INITIALIZE Statement

The INITIALIZE statement sets selected categories of data fields to predetermined values. It is functionally equivalent to one or more MOVE statements.



BY Phrase



INITIALIZE Statement - Format

Notes:

[1] IBM Extension

identifier-1
Receiving area(s).

identifier-2, literal-1
Sending area(s).

Identifier-1 can be a subscripted or reference-modified item. A complete table can be initialized if identifier-1 is a group item 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.

Note: You cannot use the INITIALIZE statement to initialize a variably located item or group that follows a DEPENDING ON phrase of an OCCURS clause within the same 01 level.

[IBM Extension]

A floating-point data item or literal can be used anywhere a numeric identifier or literal is specified.

A DBCS or national data item or literal can be used anywhere an identifier or literal is specified.

[End of IBM Extension]

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.

  [IBM Extension] A floating-point data item or floating-point literal is treated as if it is in the NUMERIC category. [End of IBM Extension]
- The same category cannot be repeated in a REPLACING phrase.
- The keyword following the word REPLACING corresponds to a category of data shown in "Classes and Categories of Data" on page 130.

When the REPLACING phrase is not used:

- [IBM Extension] SPACE is the implied sending field for alphabetic, alphanumeric, and alphanumeric-edited items. [End of IBM Extension]
- SPACE is the implied sending field for DBCS and national 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, pointer, and procedure-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
- Data items described with the FORMAT clause as DATE, TIME, or TIMESTAMP .

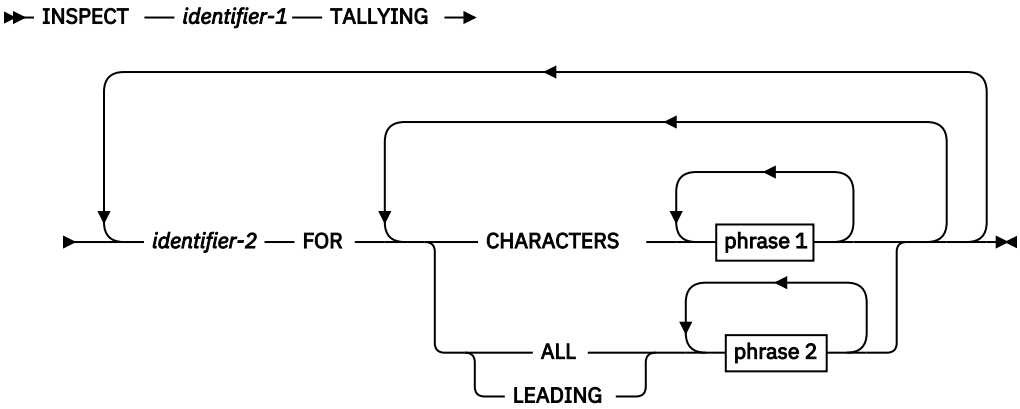
2. The areas referenced by identifier-1 are initialized in the order (left to right) of the appearance of identifier-1 in the statement. Within a group receiving field, affected elementary items are initialized in the order of their definition within the group.
3. If identifier-1 occupies the same storage 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.

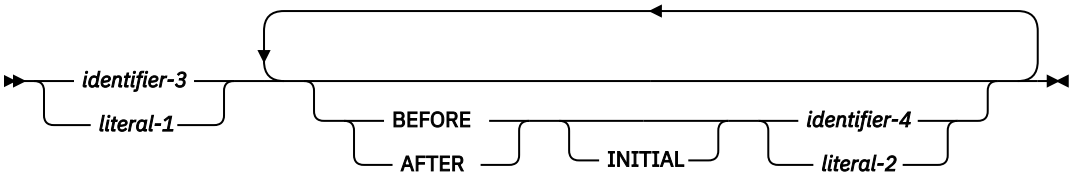
INSPECT Statement - Format 1



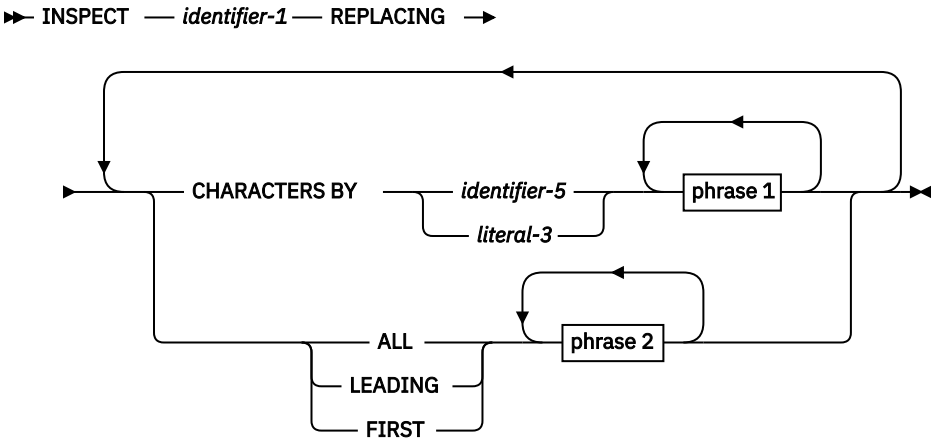
phrase 1

INSPECT Statement - Format 1

phrase 2



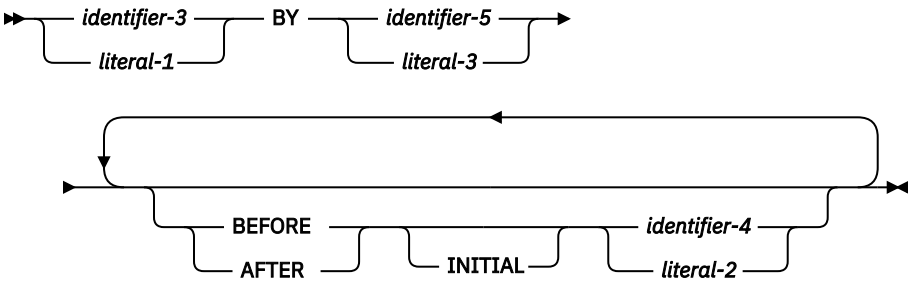
INSPECT Statement - Format 2



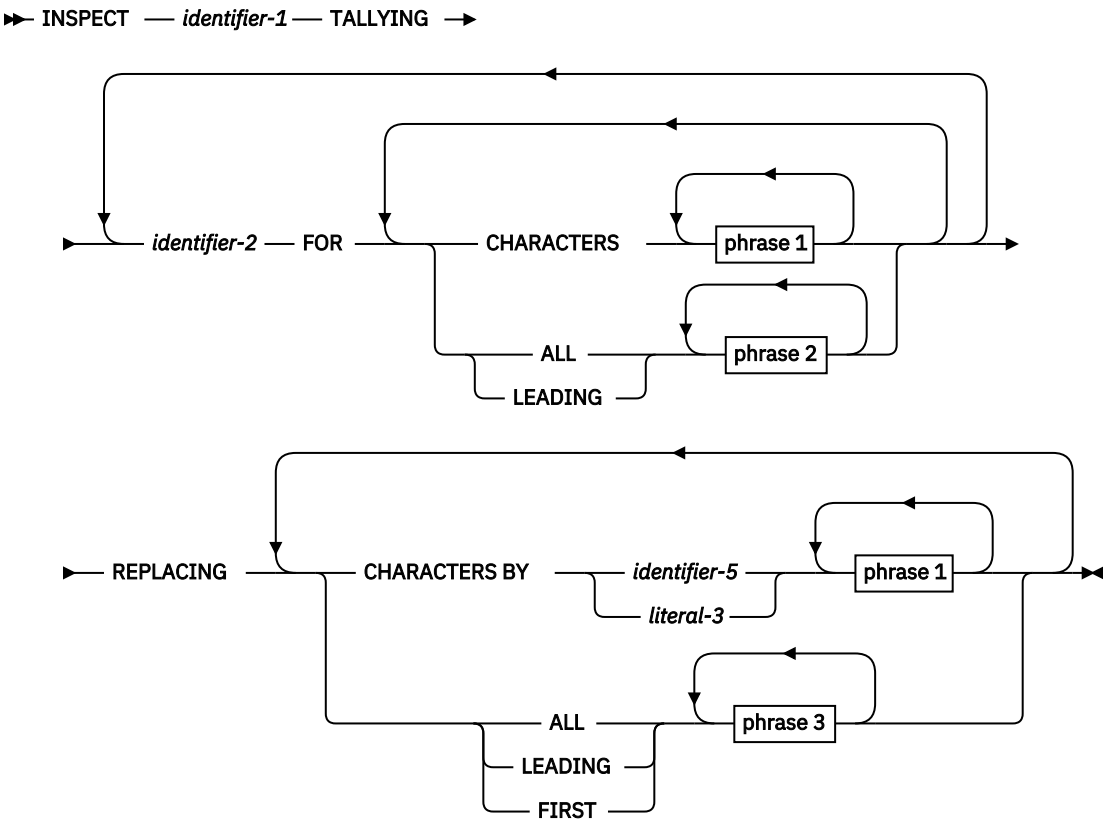
phrase 1

INSPECT Statement - Format 2

phrase 2

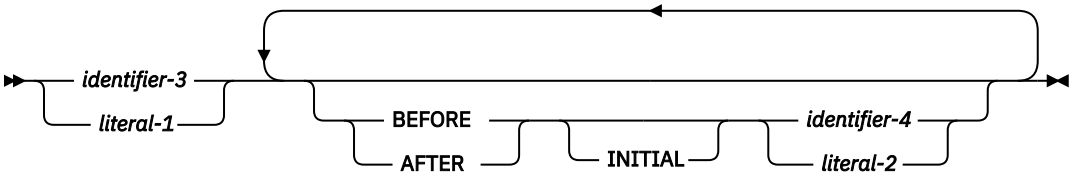


INSPECT Statement - Format 3



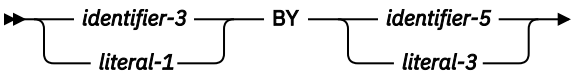
phrase 1

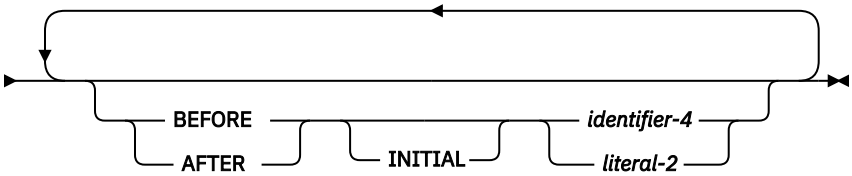
phrase 2



INSPECT Statement - Format 3

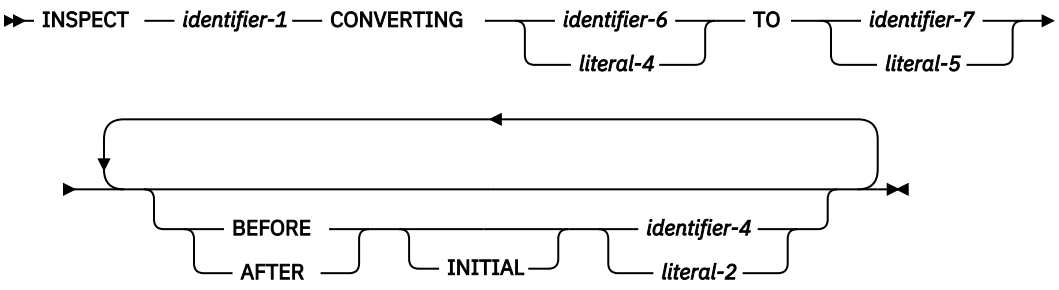
phrase 3





INSPECT Statement - Format 4

INSPECT Statement - Format 4



identifier-1
Is the inspected item; an elementary or group item with USAGE DISPLAY.

In Format-1, identifier-1 is a sending item. In the other formats, it is treated as a sending data item for the purpose of determining its length.

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) is 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.

INSPECT Statement Considerations

[IBM Extension]

If any identifiers or literals other than identifier-2 (the count field) are DBCS items, then all of them must be DBCS items.

Identifier-2 cannot be a DBCS item. DBCS characters, not bytes of data, are tallied in identifier-2.

All identifiers, except identifier-2 (the count field), can be external floating-point items. External floating-point items are treated as if redefined as alphanumeric with the INSPECT statement referring to the alphanumeric item.

[End of IBM Extension]

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, . . .)

If any identifier is subscripted, reference modified, or is a function-identifier, the subscript, reference-modifier, or function is evaluated only once as the first operation in the execution of the INSPECT statement.

Comparison Rules

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 332.

Figure 16 on page 330 is an example of INSPECT statement results.

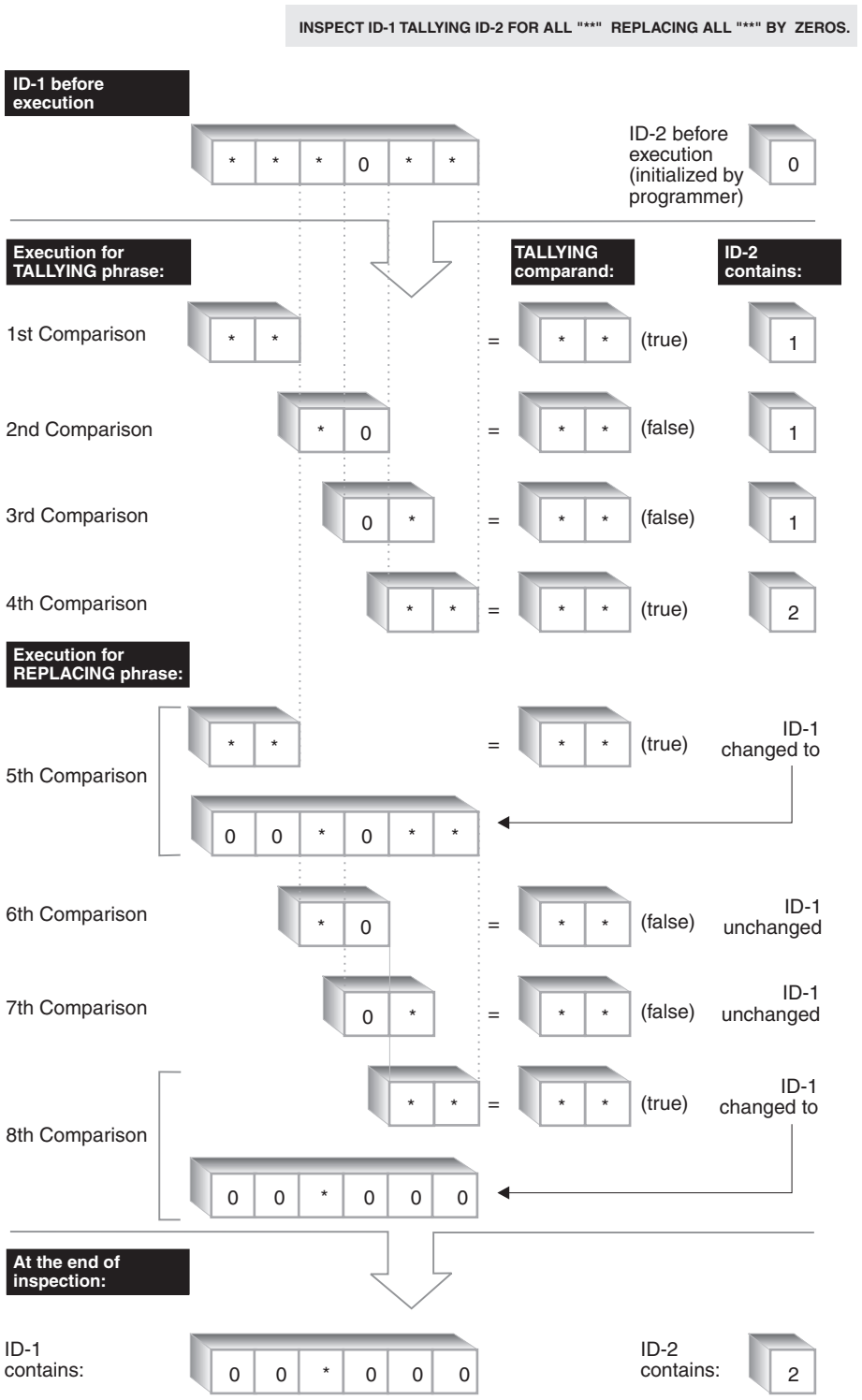


Figure 16. Example of INSPECT Statement Execution Results

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 numeric 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 (the item whose occurrences are tallied) 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 non-overlapping 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 non-overlapping 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 have 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 non-overlapping 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 non-overlapping 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, and literal-2 or identifier-4 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 non-overlapping 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 non-overlapping 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 this 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. National literals cannot be used for literal-2, literal-4, or literal-5.

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 "#".
```

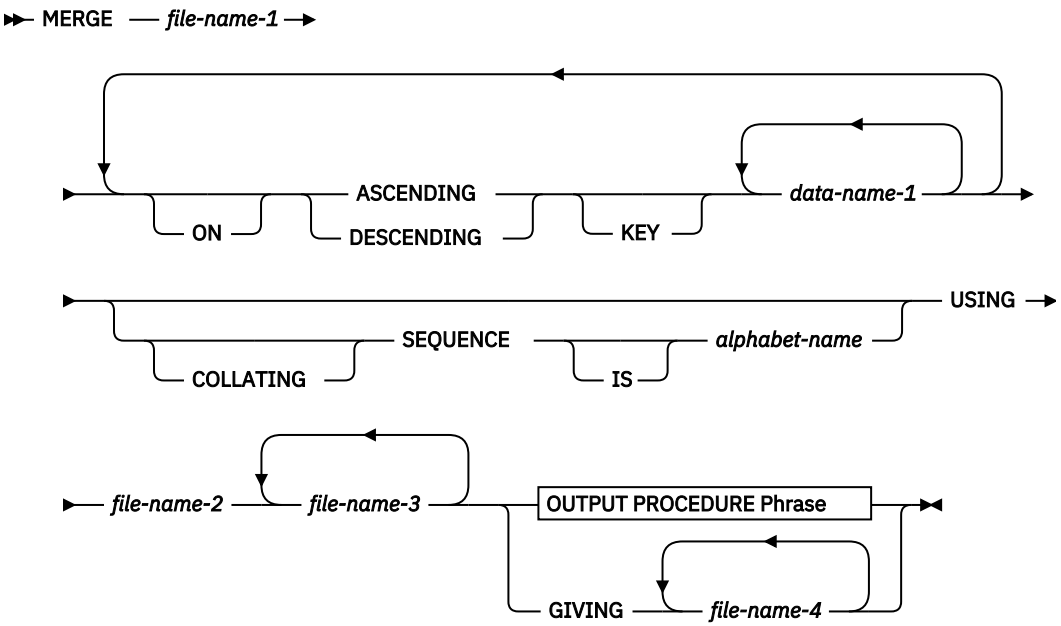
ID-1 Before	ID-1 After
AC"AEBDFBCD#AB"D	AC"XEYXFYZX#AB"D

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.

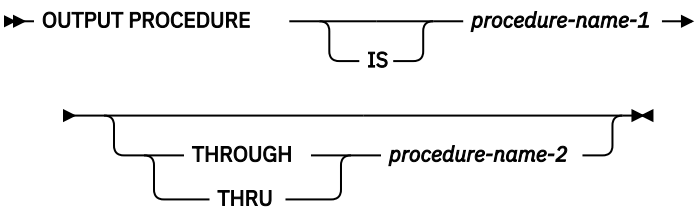
A MERGE statement may appear anywhere in the Procedure Division except in a Declarative Section. The maximum number of USING or GIVING files is 32.

[IBM Extension] It is not necessary to sequence input files prior to a merge operation. [End of IBM Extension]



MERGE Statement — Format

OUTPUT PROCEDURE Phrase



file-name-1

The name given in the SD entry that describes the record.

No file-name may be repeated in the MERGE statement.

Do not specify a pair of file names in the MERGE statement that already share storage through a SAME AREA, SAME SORT AREA, or SAME SORT-MERGE AREA clause. However, you may specify file names in the MERGE statement that share the SAME RECORD AREA clause if they are also associated with the GIVING clause (file-name-4).

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 only supported for DATABASE files that have ALWNULL specified on their ASSIGN clause.

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.

data-name-1
Is a key data-name. Records are processed in ascending or descending order on this key.

Data-name-1 specifies the KEY data item on which the merge is based. Each such data-name identifies 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.
- If file-name-1 contains variable-length records, all of the KEY data-items must be contained within the first *n* character positions of the record, where *n* equals the minimum record size specified for file-name-1.
- 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, but they cannot be subscripted or indexed.
- KEY data items cannot be variably-located.

[IBM Extension]

- KEY data items can be floating-point or date-time items.
- KEY data items can be reference modified, but they cannot be subscripted or indexed.

[End of IBM Extension]

- The total length (in bytes) of the KEY data items must not exceed 2 000
- Variable length fields can not be used in a MERGE key as a variable length field. Variable length fields are converted into group items by ILE COBOL. Since variable length fields are converted into group items, they are compared as alphanumeric data items when used in a MERGE key.

The direction of the merge operation depends on the specification of the ASCENDING or DESCENDING keywords 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 336 below). If the KEY data item is DBCS or DBCS-edited, the sequence of key values is based on a binary collating sequence of the hexadecimal values of the DBCS characters. The COLLATING SEQUENCE phrase is ignored.

[IBM Extension]

- If the KEY is an external floating-point item, the key is treated as alphanumeric. The sequence in which the records are merged depends on the collating sequence used.

- If the KEY is an internal floating-point item, the sequence of key values is in numeric order.
- If the KEY is a date-time item, only some formats will be sorted as date or time items. ILE COBOL supports many more date-time formats than IBM i DDS. In general, ILE COBOL date-time formats that match an IBM i DDS format are sorted as a date or time item; all other formats are treated as alphanumeric items, and are sorted based on their hexadecimal value.

[End of IBM Extension]

The key comparisons are performed according to the rules for comparison of operands in a relation condition (see "Relation Condition" on page 228).

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 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 NLSSORT is specified, the collating sequence is determined by the LANGID and SRTSEQ parameters of the CRTCBLMOD and CRTBNDCBL commands.
- 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

file-name-2, file-name-3, ...
Specifies input files.

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.

If file-name-1 contains variable length records, the size of the records contained in the input files must be no less than the smallest record nor greater than the largest record described for file-name-1. If file-name-1 contains fixed-length records, the size of the records contained in the input files must be no greater than the largest record described for file-name-1.

GIVING Phrase

file-name-4, ...
Specifies input files.

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:

1. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase had been executed.
2. 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.
3. The processing of the file is terminated, 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 be created without a keyed sequence access path. Otherwise, the MERGE statement cannot override the collating sequence defined in the data description specifications (DDS).

If the output files (file-name-4) contain variable-length records, the size of the records contained in file-name-1 must be no less than the largest record described in the output files. If the output files contain fixed-length records, the size of the records contained in file-name-1 must be no greater than the largest record described for the output files.

OUTPUT PROCEDURE Phrase

This phrase specifies the name of a procedure that is to select or modify output records from the merge operation.

procedure-name-1
Specifies the first (or only) section or paragraph in the OUTPUT PROCEDURE.

procedure-name-2
Identifies the last section or paragraph of the OUTPUT PROCEDURE.

The OUTPUT PROCEDURE can consist of any procedure needed to select, modify, or copy the records that are made available one at a time by the RETURN statement in merged order from the file referenced by file-name-1. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the output procedure. 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 cause the execution of any MERGE, RELEASE, or SORT statement.

If an output procedure is specified, control passes to it after the file referenced by file-name-1 has been sequenced by the MERGE statement.

Note: The OUTPUT PROCEDURE phrase is similar to a basic PERFORM statement. For example, if you name a procedure in an OUTPUT PROCEDURE, 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 is terminated after the last statement completes execution. The last statement in an OUTPUT PROCEDURE can be the EXIT statement (see "EXIT Statement" on page 318).

[IBM Extension] SORT-RETURN Special Register

The SORT-RETURN special register is the name of a binary data item and is available to both sort and merge programs.

The SORT-RETURN special register has the implicit definition:

```
01     SORT-RETURN GLOBAL PICTURE S9(4) USAGE BINARY VALUE ZERO.
```

When used in nested programs, the SORT-RETURN special register is implicitly defined as GLOBAL in the outermost COBOL program. The SORT-RETURN special register contains a return code of 0 (successful) or 16 (unsuccessful) at the completion of a sort/merge operation.

You can set the SORT-RETURN special register to 16 in an error declarative or input/output procedure to terminate a sort/merge operation before all records are processed. The operation is terminated before a record is RETURNed or RELEASEd. You may specify the SORT-RETURN special register in a function wherever an integer argument is allowed.

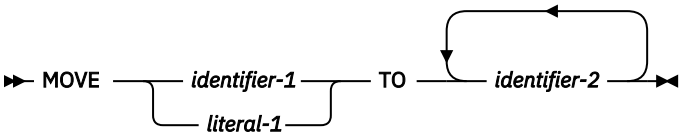
[End of IBM Extension]

MOVE Statement

The MOVE statement transfers data between areas of storage.

MOVE Statement - Format 1

MOVE Statement - Format 1



MOVE Statement - Format 2

MOVE Statement — Format 2

identifier-1, literal-1
Sending item.

identifier-2
Receiving item or items.

In Format 1, 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 identifier-2 is specified. See “Elementary Moves” on page 339 and “Group Moves” on page 344.

In Format 2, identifier-1 and identifier-2 must be group items. Selected items in identifier-1 are moved to identifier-2, according to the rules for the CORRESPONDING phrase described on page “CORRESPONDING Phrase” on page 245.

MOVE Statement Rules

[IBM Extension] If either the sending or receiving item is a DBCS data-item, then both must be DBCS items. The DBCS sending item can also be a DBCS literal or the figurative constant SPACE. No data conversion is done; the data is either truncated or padded with DBCS spaces on the right. [End of IBM Extension]

An index data item cannot be specified in a MOVE statement.

[IBM Extension] A pointer data item (USAGE POINTER) or a procedure-pointer data item (USAGE PROCEDURE-POINTER) cannot be specified in a MOVE statement. To move an address into a pointer or procedure-pointer data item, use the SET statement. [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 166).

Any length evaluation, subscripting, reference modification, or function associated with the sending item (identifier-1 or literal-1) 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 (identifier-2) 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 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 storage, in which case the contents are not predictable).

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.

[IBM Extension] De-editing also occurs for items of class date-time. In this case, all separators, and any conversion specifiers that are not numeric, are removed from the date-time item, resulting in a numeric value. [End of IBM Extension]

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]

- **Floating-point**—includes internal floating-point items (defined as USAGE COMP-1 or USAGE COMP-2), external floating-point items (defined as USAGE DISPLAY), and floating-point literals.
- **Boolean**—includes Boolean data items and Boolean literals.
- **DBCS**—includes DBCS data-items and DBCS literals.
- **National**—includes national data-items and national literals.
- **Date-Time**—includes date, time, and timestamp data items of class date-time. Date-time data items are defined as USAGE DISPLAY or PACKED-DECIMAL.

[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 131.
- 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.
- [IBM Extension] If the sending item is national, it will be converted before it is passed to the receiving field. The conversion is performed based on the data translation rule described in "National" on page 342. [End of IBM Extension]

Alphanumeric or Alphanumeric-Edited

- Alignment and any necessary space filling take place, as described under "Alignment Rules" on page 131.
- If the sending item is a national decimal integer item, the sending data is converted to usage DISPLAY and treated as though it were moved to a temporary data item of category alphanumeric with the same number of character positions as the sending item. The resulting alphanumeric data item is treated as the sending item.
- 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.
- If the sending item is national, it will be converted before it is passed to the receiving field. The conversion is performed based on the data translation rule described in "National" on page 342.
- If the sending item is date-time, the date-time item is treated like an alphanumeric item, and moved to the receiver following the rules for an alphanumeric to alphanumeric move. If the sending date-time item has a USAGE of PACKED-DECIMAL, it is first converted to a USAGE of DISPLAY.
- 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.
- If the receiving item is numeric and the sending item is alphanumeric literal, national literal, or ALL literal, then all characters of the literal must be numeric characters.

[End of IBM Extension]

Numeric or Numeric-Edited

- If the receiver is numeric, alignment by decimal point and any necessary zero filling take place, as described under "Alignment Rules" on page 131.
- 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.
- [IBM Extension] When the sending item is floating-point, the data is first converted to either a binary or internal decimal representation and is then moved. [End of IBM Extension]
- 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.

[IBM Extension]

- When the sending item is date-time, the date-time item is first de-editied. The unedited value of the date-time item is then moved to the receiving numeric or numeric-edited data item.
- If the receiver is numeric-edited, it may be specified with or without a LOCALE phrase. If the LOCALE phrase of the PICTURE clause *has not* been specified in its data description entry, the data moved to the edited data item is aligned by decimal point with zero fill or truncation at either end as required within the receiving character positions of the data item, except where editing requirements cause replacement of the leading zeroes. If the LOCALE phrase *has* been specified, alignment and zero-fill truncation take place as described in "LOCALE Phrase" on page 175.
- 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.
- If the receiving item is numeric and the sending item is alphanumeric literal, national literal, or ALL literal, then all characters of the literal must be numeric characters.

[End of IBM Extension]

[IBM Extension] Floating-Point

- The sending item is converted first to an internal floating-point item, and then moved.
- When data is moved to or from an external floating-point item, the data is converted to or from its equivalent internal floating-point value.
- It is possible that when an external floating-point literal is moved to an external floating-point data item, the external floating-point data item can receive an inaccurate value. This is because the floating-point data type is an approximation. When an external floating-point literal is moved, it is first converted to a true floating-point value (IEEE), which can also affect its accuracy.

  For example, consider the following MOVE:

```
77 external-float-1 PIC +9(3).9(13)E+9(3).
    MOVE +123455779012.3453E+297 to external-float-1.
    DISPLAY "EXTERNAL-FLOAT-1=" external-float-1.
```

  The displayed result of the MOVE is:

```
EXTERNAL-FLOAT-1=+123.4557790123452E+306
```

[End of IBM Extension]

[IBM Extension] Date-Time

- If the sending item is date-time, then the format of the sending date-time item is first converted to the receiver's format, and then moved. If the sending item is a timestamp, and the receiving item is a date or time item, then only the date or time portion of the timestamp item is moved to the receiving item. If the sending item is a date or time item and the receiving item is a timestamp, only the date or time portion of the timestamp is replaced.
- If the sending item is numeric, each of the receiving items numeric conversion specifiers are replaced with the digits from the sending item, beginning at the rightmost conversion specifier, and at the rightmost digit of that conversion specifier. All alphanumeric conversion specifiers take on default values.
- If the sending item is numeric-edited, the numeric-edited item is de-edited. The resulting numeric value is then moved to the date-time item.
- If the sending item is alphanumeric or alphanumeric-edited, the receiving date-time item is treated as an alphanumeric item, and the move takes place according to the rules for an alphanumeric to alphanumeric move.

[End of IBM Extension]

[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]

[IBM Extension] DBCS or DBCS-Edited

- If the sending item is national, it will be converted before it is passed to the receiving field. The conversion is performed based on the data translation rule described in “National” on page 342.
- Otherwise, no conversion takes place
- If the sending and receiving items are not of the same size, the data item is truncated or padded with DBCS spaces (on the right) as appropriate.

[End of IBM Extension]

[IBM Extension] National

A national data item may receive data from an alphabetic, alphanumeric, DBCS, or national data item, and also from a nonnumeric, DBCS, or national literal, or the figurative constant SPACE/SPACES.

Data moved to such an item is aligned at the leftmost character position and, where necessary, truncated or padded to the right with the padding character specified in the Padding Character command option or NTLPADCHAR option of the PROCESS statement. For information about the PROCESS statement, see *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

If the data that is being transferred is not national data, it will be converted from its representation in the sending field according to the data translation rule before it is placed in the receiving field.

The CCSID specified on the National CCSID compiler option or the NTLCCSID PROCESS option is used to define the national data CCSID.

The following rules determine the CCSID associated with other data items:

- A nonnumeric literal uses the CCSID specified for the program source file.
- A DBCS literal uses the DBCS CCSID corresponding to the CCSID specified for the program source file.
- A single-byte data item, such as alphabetic or alphanumeric, uses the CCSID specified by the second item in the CCSID options of the PROCESS statement.
- A DBCS data item uses the CCSID specified by the third item in the CCSID options of the PROCESS statement.

[End of IBM Extension]

Valid Elementary Moves

Table 32 on page 343 shows valid and invalid elementary moves for each category. In the table:

- YES = Move is valid.
- NO = Move is invalid

Table 32. Valid Elementary Moves

Sending Item Category	Receiving Item Category										
	Alphabetic	**Alphanumeric, Alphanumeric-edited**	**Numeric, Numeric-edited**	**BOOLEAN (6)**	**DBCS (8)**	**External Floating-Point (6)**	**Internal Floating-Point (6)**	**Date (6)**	**Time (6)**	**Timestamp (6)**	**National (6)**
Alphabetic and SPACE	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	YES
Alphanumeric (1)	YES	YES	YES	YES (5)	NO	YES (9)	YES (9)	YES	YES	YES	YES
Alphanumeric-edited	YES	YES	NO	NO	NO	NO	NO	YES	YES	YES	NO
Numeric Integer (2)	NO	YES	YES	NO	NO	YES	YES	YES	YES	YES	YES
Numeric Noninteger (3)	NO	NO	YES	NO	NO	YES	YES	NO	NO	NO	NO
Numeric-edited	NO	YES	YES	NO	NO	YES	YES	YES	YES	YES	NO
LOW/HIGH-VALUE, QUOTES	NO	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO
ZERO	NO	YES	YES	YES	NO	YES	YES	NO	NO	NO	YES
BOOLEAN (4) (6)	NO	YES	NO	YES	NO	NO	NO	NO	NO	NO	NO
DBCS (6) (7) (8) DBCS-edited	NO	NO	NO	NO	YES	NO	NO	NO	NO	NO	YES
Floating-Point (10)	NO	NO	YES	NO	NO	YES	YES	NO	NO	NO	NO
Date (6)	NO	YES	YES	NO	NO	NO	NO	YES	NO	YES	NO
Time (6)	NO	YES	YES	NO	NO	NO	NO	NO	YES	YES	NO
Timestamp (6)	NO	YES	YES	NO	NO	NO	NO	YES	YES	YES	NO
National (6)	YES	YES	YES (11)	NO	YES	NO	NO	NO	NO	NO	YES

Notes to Table 32 on page 343:

(1)
Includes nonnumeric literals

(2)
Includes integer numeric literals

(3)
Includes noninteger numeric literals

(4)
Includes Boolean literals

(5)
First character of sending item is moved, regardless of its value

(6)
Boolean, DBCS, DBCS-edited, national, internal and external floating-point, and date-time items are an IBM Extension.

(7)
Includes DBCS literals and SPACE.

(8)
Includes DBCS data-items.

(9)
Figurative constants and nonnumeric literals must consist only of numeric characters and will be treated as numeric integer fields. The ALL literal may not be used as a sending item.

(10)
[IBM Extension] Includes floating-point literals, external floating-point data items (USAGE DISPLAY), and internal floating-point data items (USAGE COMP-1 or USAGE COMP-2). [End of IBM Extension]

(11)
National data-items can be moved to numeric but not to numeric-edited.

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.

[IBM Extension]

In the following discussion, on the MOVE statement and pointers, *pointers* refers to both the pointer data item (USAGE POINTER) and the procedure-pointer data item (USAGE PROCEDURE-POINTER).

A pointer can be part of a group that is referred to in a MOVE statement.

A pointer move occurs when all of the following conditions are met:

- The sending or receiving item of a MOVE statement contains a pointer
- Both items are at least 16 bytes long and properly aligned
- 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 210.

A pointer can be part of a group that is referred to in a MOVE CORRESPONDING statement; however, movement of the pointer will not take place.

[End of IBM Extension]

[IBM Extension] WHEN-COMPILED Special Register

This special register contains the date at the start of compilation. It consists of an alphanumeric data item with the implicit definition:

```
01     WHEN-COMPILED GLOBAL PICTURE X(16) USAGE DISPLAY
```

and format:

```
MM/DD/YYhh.mm.ss (MONTH/DAY/YEARhour.minute.second)
```

For example, if compilation began at 2:04 PM on 15 December 1994, WHEN-COMPILED would contain the value 12/15/9414.04.00.

The DATSEP or TIMSEP parameter of job-related commands (such as CHGJOB) specifies the date-separation or time-separation character used in the WHEN-COMPILED special register. The DATFMT parameter specifies the date format used in the WHEN-COMPILED special register.

It is valid only as the sending item in a MOVE statement.

The special register data can be reference-modified only when it is used as a sender data item.

In nested programs, this special register is implicitly defined in the outermost program.

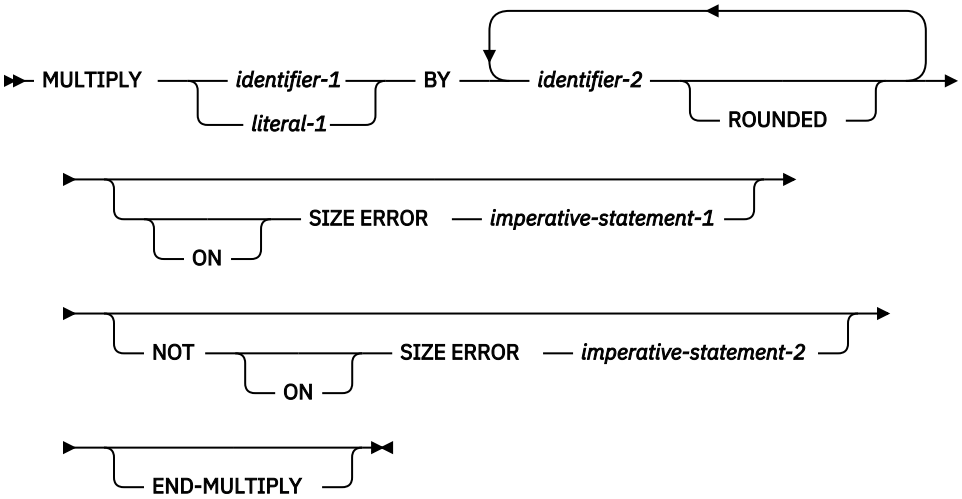
Note: The compilation date and time is also accessible using the date and time intrinsic function WHEN-COMPILED (see "WHEN-COMPILED" on page 503). That function supports 4-digit year values, and provides additional information.

[End of IBM Extension]

MULTIPLY Statement

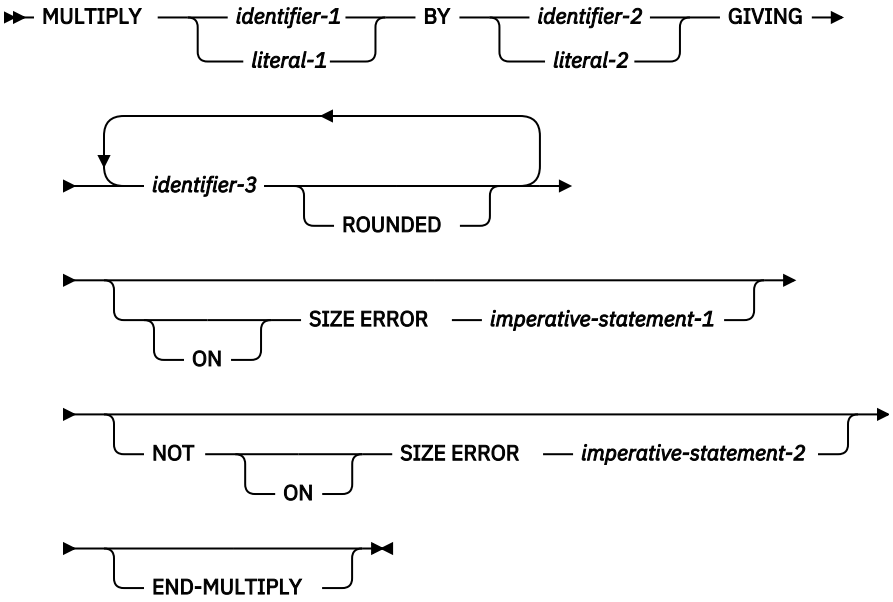
The MULTIPLY statement multiplies numeric items and stores the result.

MULTIPLY Statement - Format 1



In Format 1, the value of identifier-1 or literal-1 is saved. This value is multiplied by and stored in each identifier-2, in the left-to-right order in which identifier-2 is specified.

MULTIPLY Statement - Format 2 - GIVING



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 each data item referenced by identifier-3.

For all Formats:

identifier-1, identifier-2
Must be an elementary numeric item.

literal
Must be a numeric literal.

identifier-3
Must be an elementary numeric or numeric-edited item.

The composite of operands is determined by superimposing all of the receiving data items. For more information on the composite of operands, see the "Size of Operands" on page 247.

[IBM Extension] Floating-point data items and literals can be used anywhere a numeric data item or literal can be specified. [End of IBM Extension]

Note: Intermediate results generated during the execution of a MULTIPLY statement are system-specific and can affect program portability.

ROUNDED Phrase

For Formats 1 and 2, see "ROUNDED Phrase" on page 246.

SIZE ERROR Phrases

For Formats 1 and 2, see "SIZE ERROR Phrases" on page 246.

END-MULTIPLY Phrase

This explicit scope terminator serves to delimit the scope of the MULTIPLY statement. END-MULTIPLY converts a conditional MULTIPLY statement into an imperative statement. This allows it to be nested in another conditional statement.

For more information, see "Delimited Scope Statements" on page 243.

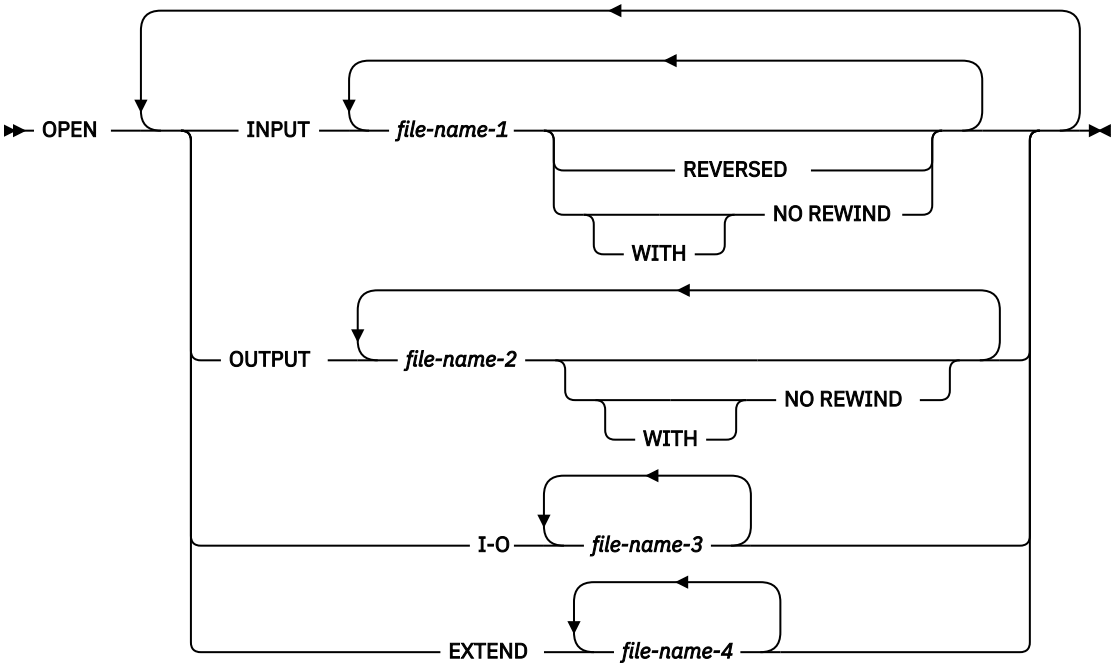
OPEN Statement

The OPEN statement initiates the processing of files and checks or writes labels.

The OPEN statement varies depending on the type of file.

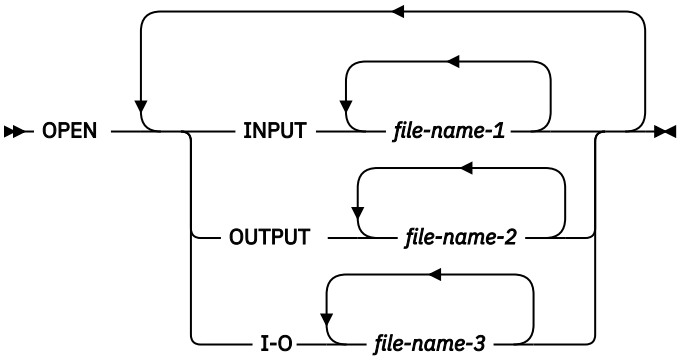
OPEN Statement - Format 1 - Sequential

OPEN Statement — Format 1 — Sequential



OPEN Statement - Format 2 - Indexed and Relative

OPEN Statement - Format 2 - Indexed and Relative



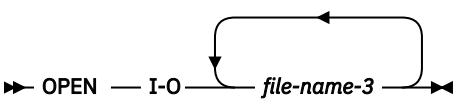
For relative files only, the OPEN statement is not allowed for logical file members:

- That are based on more than one physical file
- That contain select or omit logic

OPEN Statement - Format 3 - TRANSACTION

[IBM Extension]

OPEN Statement - Format 3 - TRANSACTION



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 274.

[End of IBM Extension]

INPUT
Permits opening the file for input operations.

Not allowed for FORMATFILE or printer files.

OUTPUT
Permits opening the file for output operations. This phrase will cause sequential and relative DISK files to be dynamically created if they do not exist and CRTF option is specified. When a file is opened OUTPUT it contains no records.

Existing records are removed (cleared) only for physical files. For logical files, the file is treated as though EXTEND had been specified.

I-O
Permits opening the file for both input and output operations. The I-O phrase can be specified only for files assigned to direct access devices, such as DISK, DATABASE, and workstation files.

EXTEND
Permits opening the file for output operations.

The EXTEND phrase must not be specified for a multiple file reel.

The EXTEND phrase is not allowed for:

- FORMATFILE files
- Printer files
- DISKETTE files

file-name-1, file-name-2, file-name-3, file-name-4
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.

REVERSED
Valid only for sequential single reel tape files.

NO REWIND
Valid only for sequential single reel tape files.

OPEN Statement Considerations

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. "OPEN Statement Programming Notes" on page 352 shows the results of opening available and unavailable files.

Dynamic File Creation

In some cases, a file that would not otherwise be available will be created by the OPEN statement. This feature is referred to as *Dynamic File Creation*.

[IBM Extension]

In ILE COBOL, dynamic file creation will only occur for files that are assigned to DISK. In addition, either OPTION(*CRTF) must be specified in the CRTCBLMOD or CRTBNDCBL command, or the CRTF option must be included in a PROCESS statement. If OPTION(*NOCRTF) or PROCESS NOCRTF is specified, or if the option is not defined, then no file that is defined in the program can be created dynamically.

If dynamic file creation has been specified, the following types of file will be created if they are not present when the OPEN statement is executed:

- Sequential and Relative files opened for OUTPUT.
- Optional Sequential and Relative files opened for I-O.
- Optional Sequential files opened EXTEND.

Optional files are those defined using a SELECT OPTIONAL clause. A compile-time error message will be issued for an OPEN I-O or OPEN EXTEND statement for an optional file, unless dynamic file creation is in effect for the file.

The default attributes of a dynamically created file are based on those of the file QAXXDBF held in library QSYS. The command CHGPF may be used to change these attributes, for example, to increase the maximum number of records, or to reduce the record wait time.

If a library-name has been provided by means of a file override, the file will be created in that library. If no file override is in effect, the file will be created in the current library, or if no current library is defined, in library QTEMP.

The maximum record length for a file that can be created dynamically is 32 766 characters.

[End of IBM Extension]

[IBM Extension] Special Considerations for Device Type DATABASE

The file may be placed under commitment control. See "Commitment Control" in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information.

If the file contains null-capable fields, and ALWNULL was not specified on the ASSIGN clause for the file, a file status of 0P is set, and only the records that do not contain null fields may be processed. Otherwise, if ALWNULL was specified for a null-capable file, null fields can be processed, and file status 0P is not set.

[End of IBM Extension]

INPUT Phrase (Sequential Files)

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.

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.

Under the OPTION(*NOBLK) option, 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 and the file is open only for input or output.
- The file is assigned to DISK, DATABASE, DISKETTE, or TAPEFILE.

The BLOCK CONTAINS clause does not control the blocking factor for any files except tape files. The BLOCK CONTAINS clause controls the blocking factor for all files.

Special Considerations for Device Types DATABASE, TAPEFILE, and DISKETTE

If SELECT OPTIONAL is specified in the file-control entry and OPTION(*CRTF) is specified in the CRTCBLMOD or CRTBNDCBL command, this combination is not valid.

[IBM Extension] Special Considerations for Device Types DISK and DATABASE

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. For more information on this command, see the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

[End of IBM Extension]

OUTPUT Phrase (Sequential Files)

The file is opened to allow only output operations. When the file is successfully opened, it contains no records.

Under 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 and the file is open only for input or output.
- The file is assigned to DISK, DATABASE, DISKETTE, or TAPEFILE.

The BLOCK CONTAINS clause does not control the blocking factor for any files except tape files.

Device type FORMATFILE and PRINTER can only be opened for output.

Special Considerations for Device Type DISK

If dynamic file creation has been specified, then the OPEN statement will create the file if it is not already available.

[IBM Extension] Special Considerations for Device Types DISK, DATABASE, and FORMATFILE

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]

I-O Phrase (Sequential Files)

Only device types DISK and DATABASE can be opened for I-O.

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.

[IBM Extension] 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. For more information on this command, see the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/. [End of IBM Extension]

Special Considerations for Device Type DISK

If dynamic file creation has been specified and the file is an optional file (SELECT OPTIONAL in the file-control entry), then the OPEN statement will create the file if it is not already available.

NO REWIND Phrase (Sequential Files)

This phrase applies only to device type TAPEFILE.

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 multifile 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]

REVERSED Phrase (Sequential Files)

This phrase applies only to device type TAPEFILE.

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.

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.

EXTEND Phrase (Sequential Files)

Device types TAPEFILE, DISK, and DATABASE may be opened as EXTEND.

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.

Special Considerations for Device Type DISK

If dynamic file creation has been specified and the file is an optional file (SELECT OPTIONAL in the file-control entry), then the OPEN statement will create the file if it is not already available.

INPUT Phrase (Indexed and Relative Files)

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.

Special Considerations for Sequential Access Mode

[IBM Extension] 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. For more information on this command, see the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/. [End of IBM Extension]

Under 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 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
- No START statements are specified for the file.

The BLOCK CONTAINS clause does not control the blocking factor.

START statements are allowed if you specify both OPTION(*BLK) and the BLOCK CONTAINS clause. The BLOCK CONTAINS clause controls the blocking factor for all files.

Special Considerations for Dynamic Access Mode

[IBM Extension] 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 *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information on this command. [End of IBM Extension]

Also, under 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 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.

OUTPUT Phrase (Indexed and Relative Files)

[IBM Extension] 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. To clear a logical file, all the members on which the logical file is based should be cleared. [End of IBM Extension]

Special Considerations for Relative Files—Device Type DISK

If dynamic file creation has been specified, then the OPEN statement will create the file if it is not already available.

Special Considerations for Indexed Files—Sequential Access

Under 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 indexed and the file is open only for input or output.
- The file is assigned to DISK or DATABASE
- No START statements are specified for the file.

The BLOCK CONTAINS clause does not control the blocking factor.

If you specify both OPTION(*BLK) and the BLOCK CONTAINS clause, the blocking factor applies.

Special Considerations for Indexed Files—Dynamic Access

Under 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 indexed and the file is open only for input or output.
- 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.

I-O Phrase (Indexed and Relative Files)

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.

Special Considerations for Relative Files—Device Type DISK

If dynamic file creation has been specified and the file is an optional file (SELECT OPTIONAL in the file-control entry), then the OPEN statement will create the file if it is not already available.

[IBM Extension] Special Considerations for Sequential or Dynamic Access Modes

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 *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information on this command.

[End of IBM Extension]

OPEN Statement Programming Notes

The successful execution of an OPEN statement determines the availability of the files and results in that file being in open mode. Table 33 on page 353 summarizes the results of opening available and unavailable files.

Table 33. 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 may cause the file to be created(1)
OUTPUT	Normal open; the file contains no records	Open may cause the file to be created(1)
EXTEND	Normal open	Open is unsuccessful
EXTEND (optional file)	Normal open	Open may cause the file to be created(1)
Note: (1) If dynamic file creation has been specified, and if the file has the appropriate organization, the file will be created. See "Dynamic File Creation" on page 348.		

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.
3. The READ statement is executed on a file which is open for INPUT or I-O.
4. The WRITE statement is executed on a file which is open for OUTPUT or EXTEND (sequential files only). The WRITE statement is also executed on an indexed or relative file which is open for I-O in random or dynamic access mode, and on a TRANSACTION file open for I-O.
5. The REWRITE statement is executed on a file which is open for I-O.
6. The START statement is executed on an indexed or relative file which is open for INPUT or I-O.
7. The DELETE statement is executed on an indexed or relative file which is open for I-O.
8. 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.
9. 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 250.
10. 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.

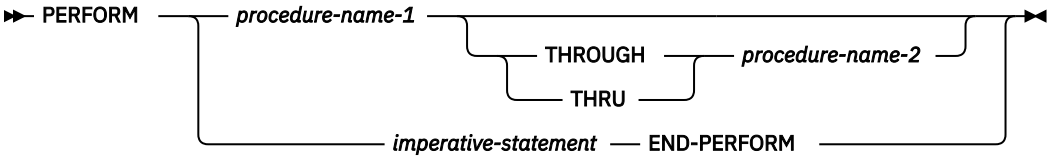
There are four PERFORM statement formats:

- Format 1 - Basic PERFORM
- Format 2 - PERFORM with TIMES Phrase
- Format 3 - PERFORM with UNTIL Phrase
- Format 4 - PERFORM with VARYING Phrase

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.

PERFORM Statement - Format 1



procedure-name
Must be 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 the PERFORM statement is in a declarative section, procedure-name-1 and procedure-name-2 must also be in a declarative section.

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.

In-line PERFORM Statement

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.

Out-of-line PERFORM Statement

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.

Nested PERFORM Statements

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.

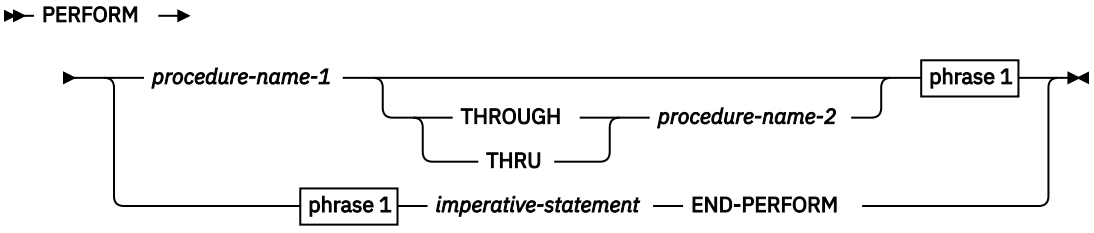
When the performed procedures include another PERFORM statement, the sequence of procedures associated with the embedded PERFORM statement must be totally included in or totally excluded from the performed procedures of the first PERFORM statement. That is, an active PERFORM statement whose execution 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]

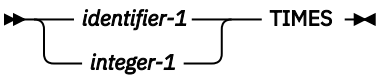
When control passes to the sequence of procedures by means other than a PERFORM statement, control passes through the exit point to the next executable statement, as if no PERFORM statement referred to these procedures.

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.



PERFORM Statement - Format 2
phrase-1



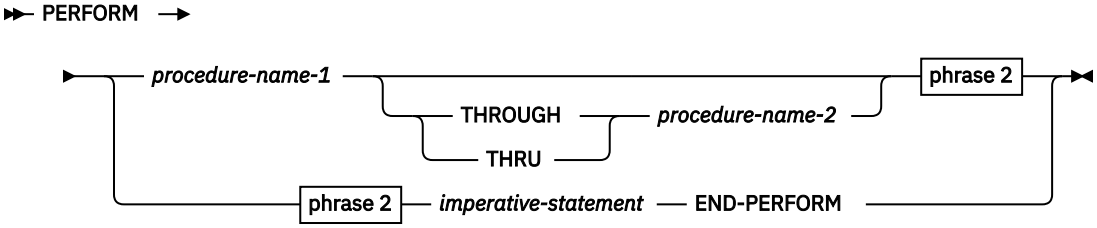
identifier-1
Must be an integer item.

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.

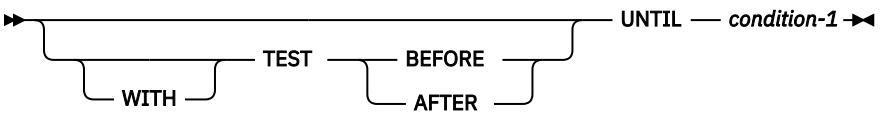
After the PERFORM statement has been initiated, any change to identifier-1 has no effect in varying the number of times the procedures are initiated.

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.



PERFORM Statement - Format 3
phrase 2



condition-1
May be any condition described under “Conditional Expressions” on page 225. If the condition is true at the time the PERFORM statement is initiated, the specified procedure(s) are not executed.

Any subscripting, reference modifier, or function 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).

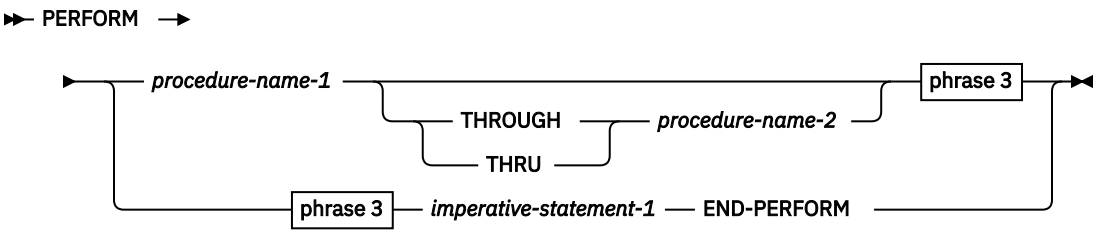
If the TEST AFTER phrase is specified, the statements to be performed are executed at least once before the condition is tested (corresponds to DO UNTIL).

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.

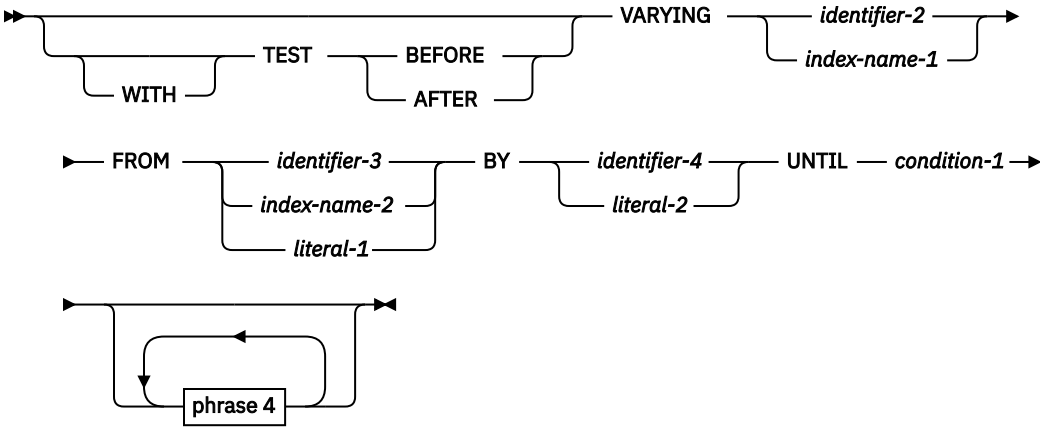
PERFORM with VARYING Phrase

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 363.)

The Format 4 VARYING phrase PERFORM statement can serially search an entire 7-dimensional table.

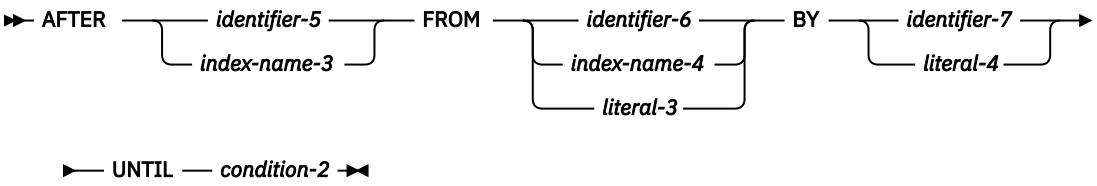


phrase 3



PERFORM Statement - Format 4

phrase 4



condition-1, condition-2

May be any condition described under “Conditional Expressions” on page 225. 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.

If any of the operands specified in condition-1 or condition-2 is subscripted, reference modified, or is a function-identifier, the subscript, reference-modifier, or function is evaluated each time the condition is tested.

[IBM Extension]

Floating-point data items and literals can be used anywhere a numeric data item or literal can be specified.

[End of IBM Extension]

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

This section provides an example that describes how to use the PERFORM statement to vary 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-1).
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) is executed once.
4. **Identifier-2** is augmented by identifier-4 (or literal-2), and condition-1 is evaluated again.
5. Steps "2" on page 358 through "4" on page 358 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 17 on page 358 illustrates the logic of the PERFORM statement when an identifier is varied with TEST BEFORE. Figure 18 on page 359 illustrates the logic of the PERFORM statement when an identifier is varied with TEST AFTER.

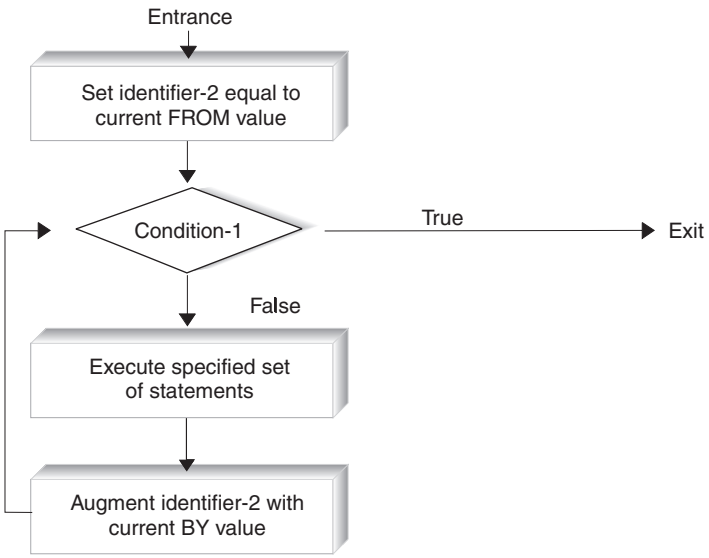


Figure 17. Varying One Identifier—with TEST BEFORE

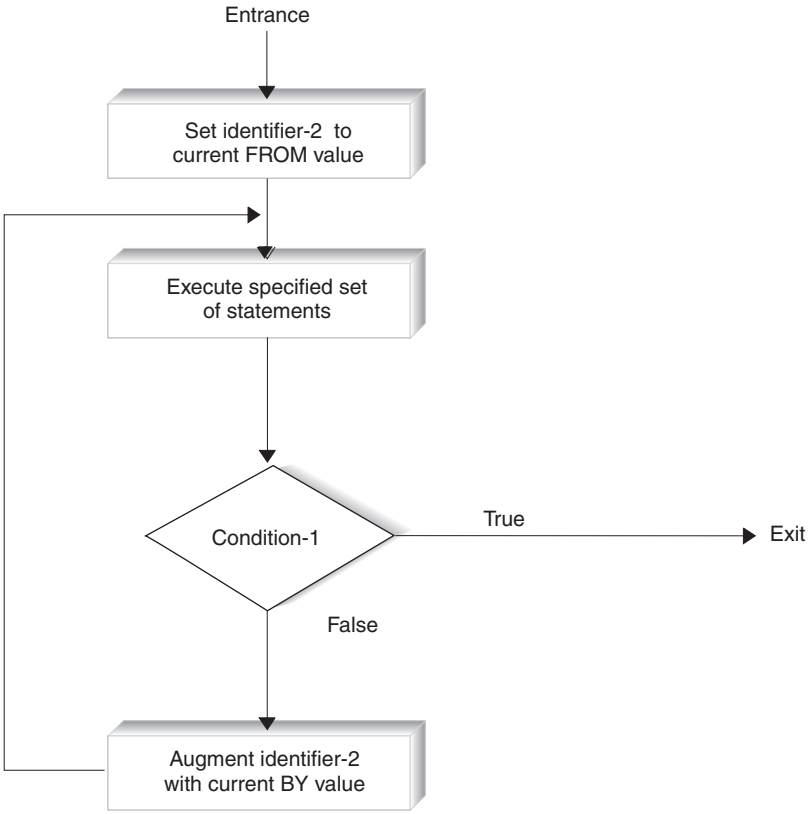


Figure 18. Varying One Identifier—with TEST AFTER

Varying Two Identifiers

This section provides an example that describes how to use the PERFORM statement to vary 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" on page 359 through "5" on page 359 are repeated until condition-2 is true.
7. Steps "2" on page 359 through "6" on page 359 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 19 on page 360 illustrates the logic of the PERFORM statement when two identifiers are varied with TEST BEFORE. Figure 20 on page 361 illustrates the logic of the PERFORM statement when two identifiers are varied with TEST AFTER.

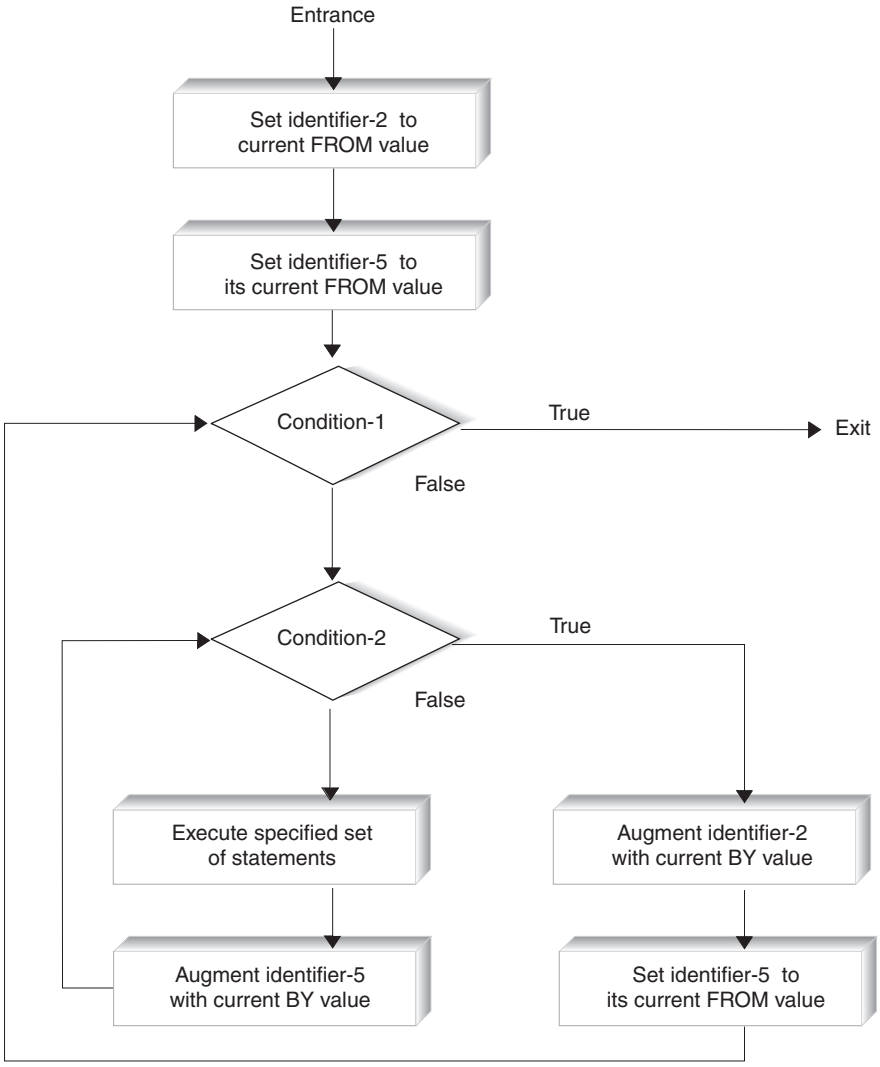


Figure 19. Varying Two Identifiers—with TEST BEFORE

The previous figure assumes that identifier-5 and identifier-2 are not related. If one is dependent on the other (through subscripting, for example), the results may be predictable but generally undesirable.

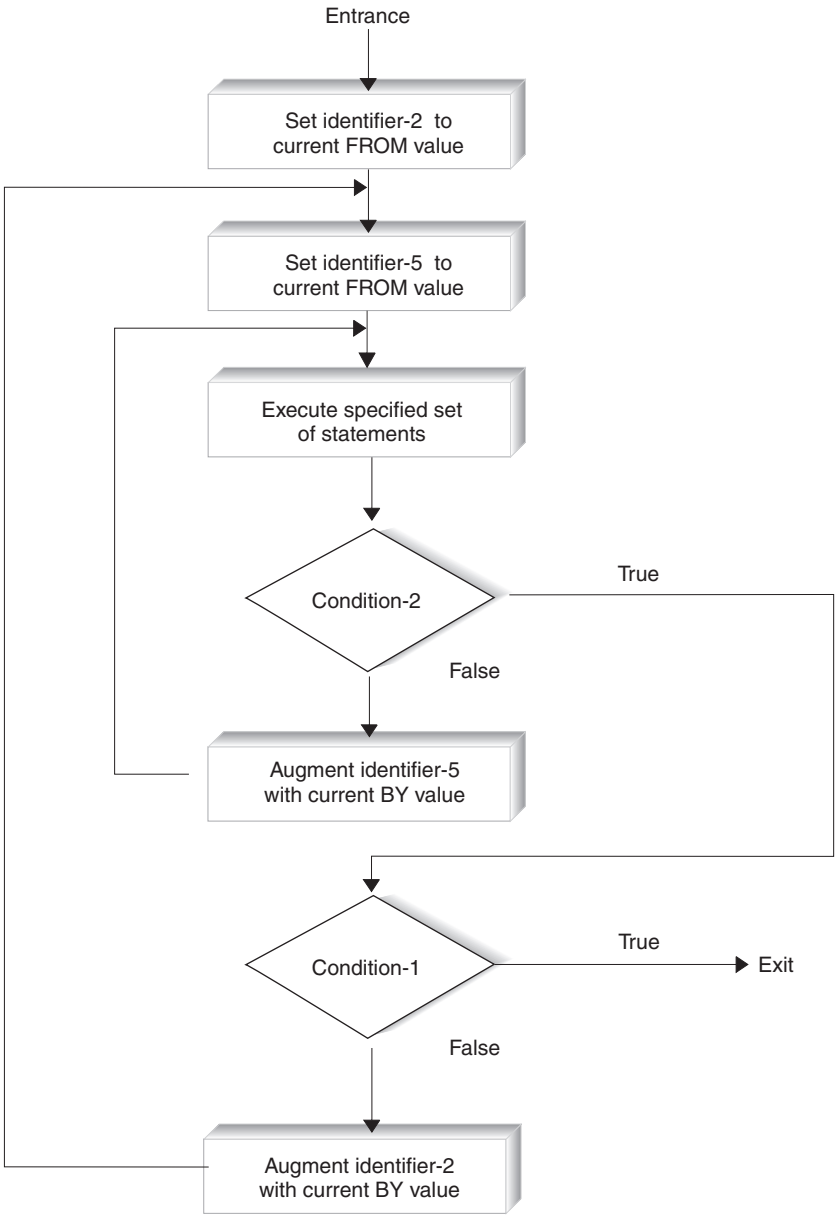


Figure 20. Varying Two Identifiers—with TEST AFTER

Varying Three Identifiers

This section provides an example that describes how to use the PERFORM statement to vary 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 21 on page 362 illustrates the logic of the PERFORM statement when three identifiers are varied.

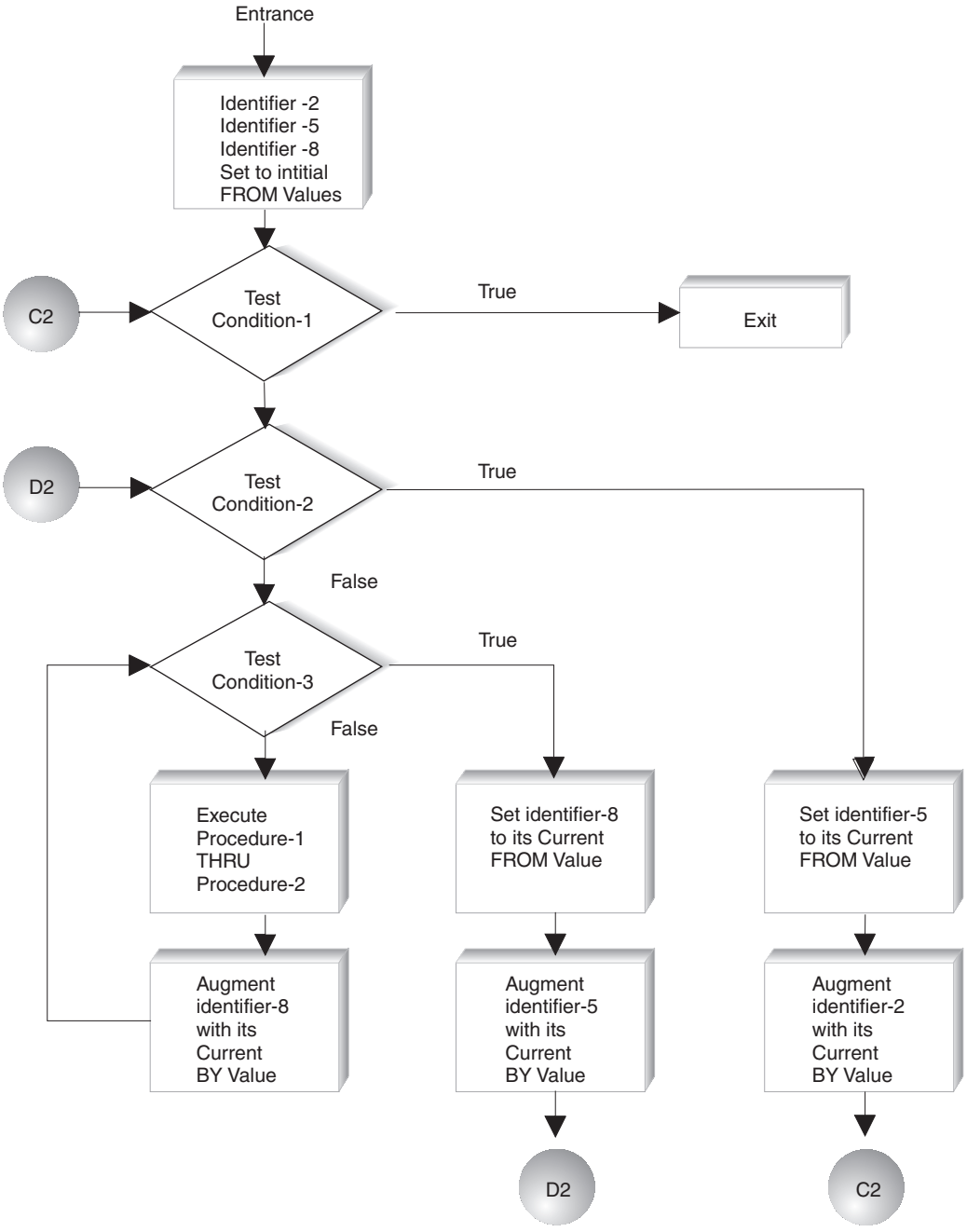


Figure 21. Format 4 PERFORM Statement Logic—Varying Three Identifiers

The previous figure assumes that identifier-5 and identifier-2 are not related. If one is dependent on the other (through subscripting, for example), the results may be predictable but generally undesirable.

The previous figure also assumes that identifier-8 and identifier-5 are not related. If one is dependent on the other (through subscripting, for example), the results may be predictable but generally undesirable.

Varying More Than Three Identifiers

In the VARYING phrase, you may extend the previous examples 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 170. (See also "SET Statement" on page 395.)
   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 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. Execution of the READ statement depends on the file organization. File organization can be:

- Sequential
- Relative
- Indexed

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.

[IBM Extension] Special Considerations for Device Types DISK and DATABASE

Null-capable fields are supported when the READ statement is performed on a file which is found on a DISK or DATABASE device. However, null values are only supported for DATABASE files that have ALWNULL specified on their ASSIGN clause. If ALWNULL is not specified, the READ operation will fail and file status 90 will be returned, if a field contains a null value. You should also specify NULL-MAP/NULL-KEY-MAP on your READ statement, so you can see which fields contain the null value.

[End of IBM Extension]

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 record from the file. The next record accessed is determined by the file organization.

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.

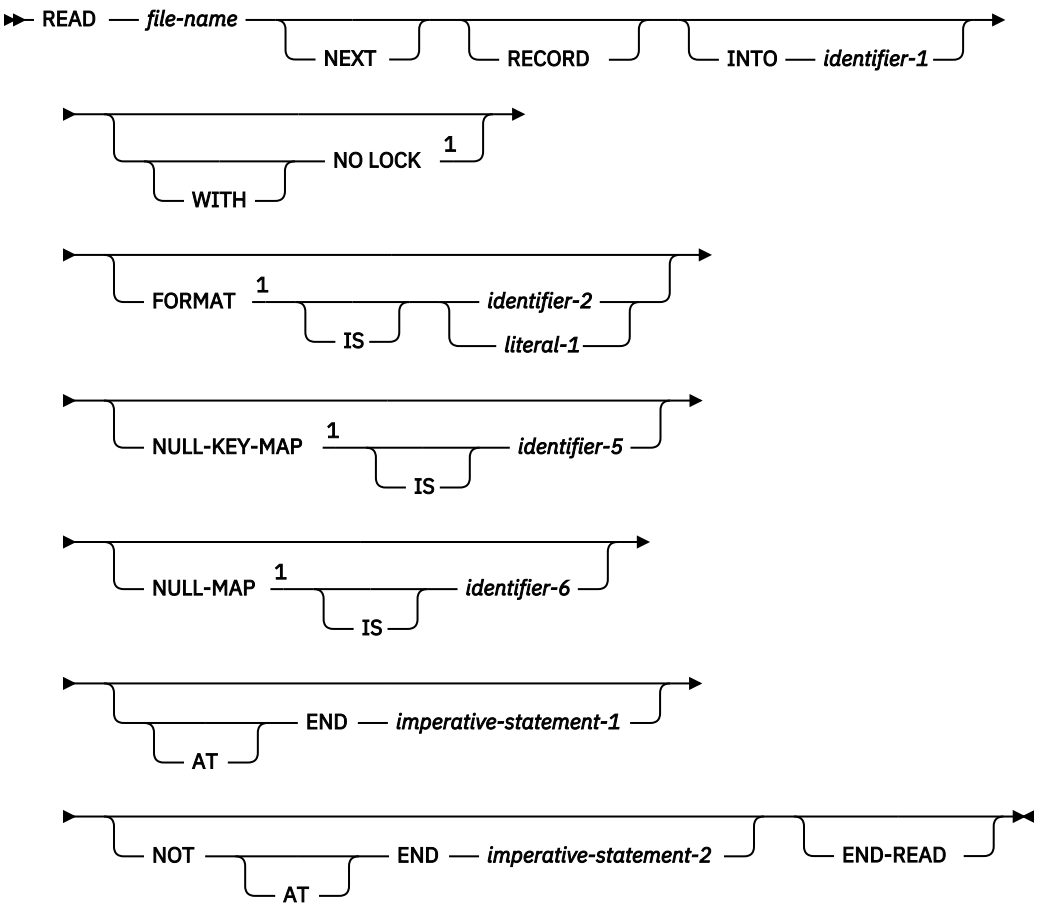
Format 3 must be specified for random retrieval. All other rules for random access apply.

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.

READ Statement - Format 1 - Sequential Retrieval/Sequential Access

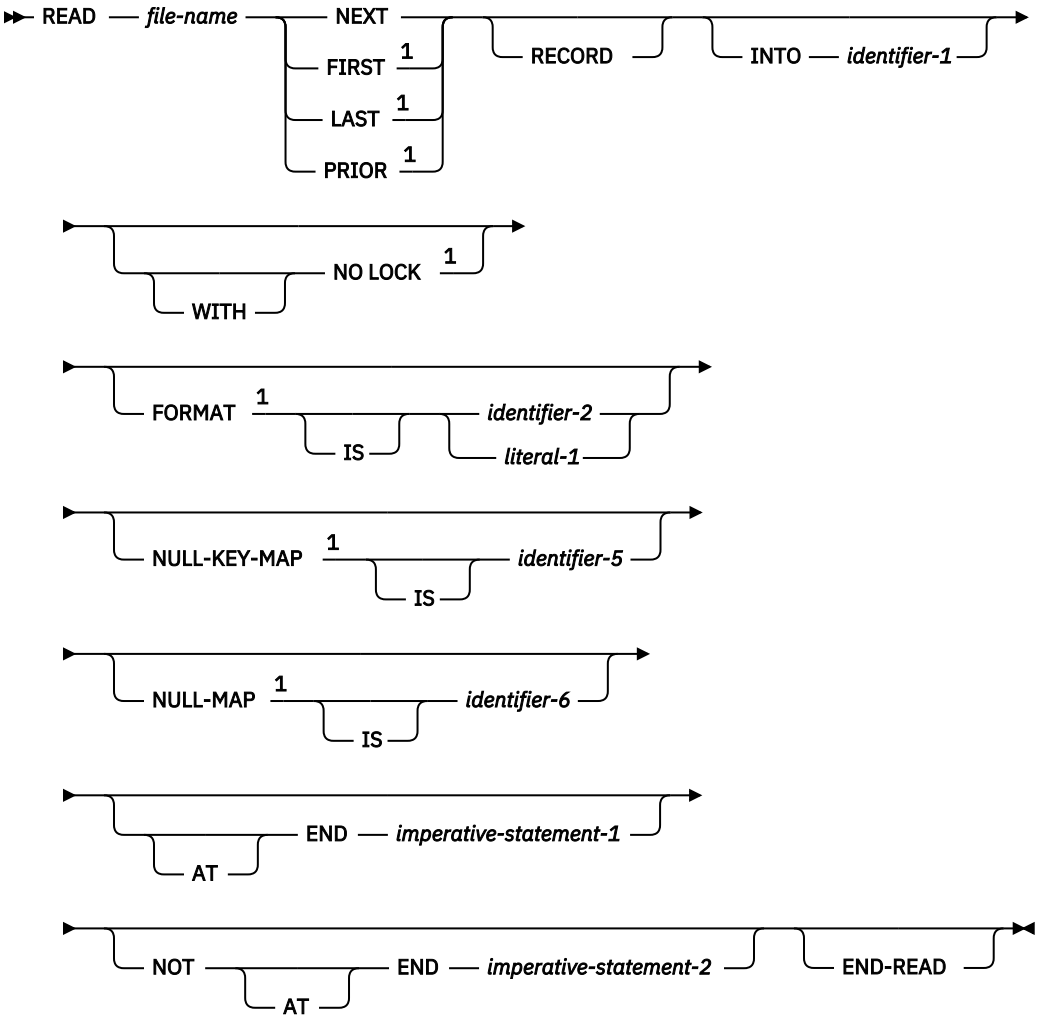


READ - Format 1 - Sequential Retrieval/Access

Notes:

[1] IBM Extension.

READ Statement - Format 2 - Sequential Retrieval/Dynamic Access

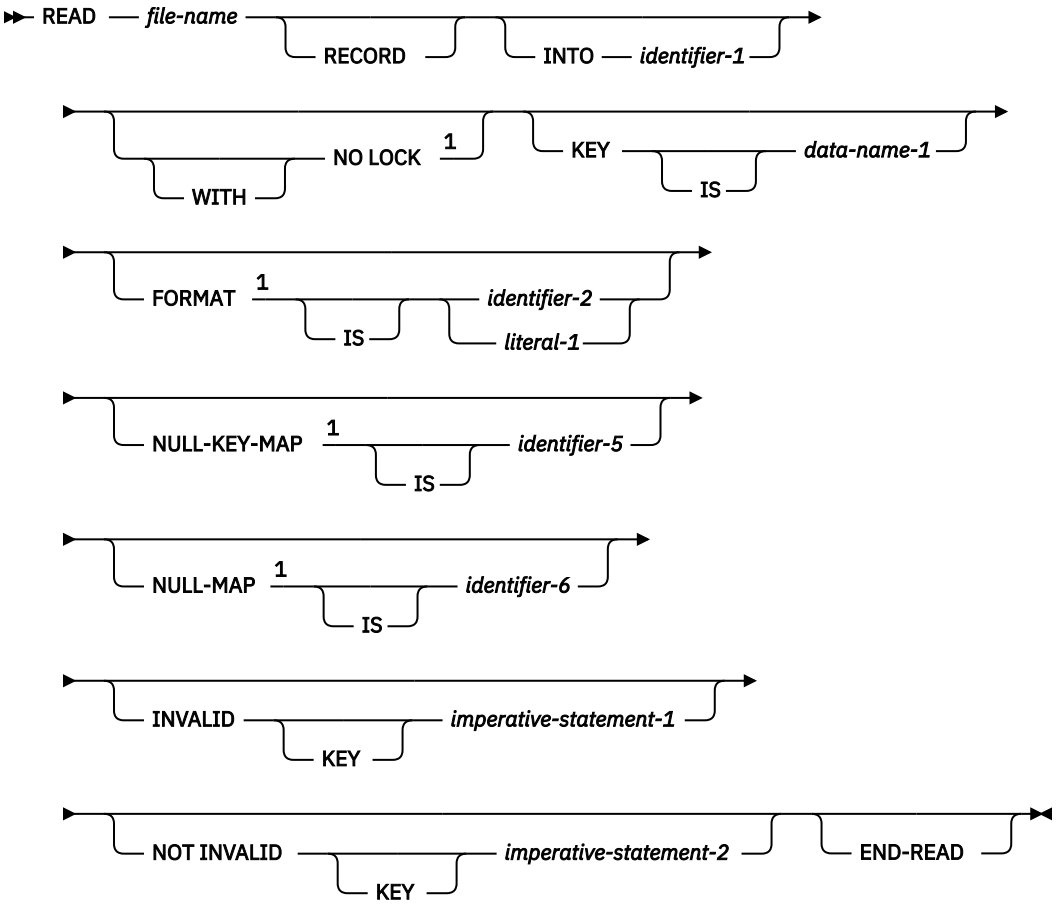


READ - Format 2 - Sequential Ret./Dynamic Access

Notes:

[1] IBM Extension

READ Statement - Format 3 - Random Retrieval



READ Statement - Format 3 - Random Retrieval

Notes:

[1] IBM Extension

file-name

File-name must be defined in a Data Division FD entry, and must not name a sort or merge file.

RECORD

The next record in the sequence of records.

KEY IS data-name-1

This phrase is specified only for indexed files. Data-name-1 can be qualified; it cannot be subscripted. Data-name-1 must identify a record key associated with file-name.

[IBM Extension] Data-name-1 can be defined as a DBCS data-item. When the RECORD KEY clause specifies a DBCS data-item, a KEY specified on the READ statement must be a DBCS data-item. [End of IBM Extension]

INTO Phrase

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 250.)

The INTO phrase may be specified in a READ statement 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 or an elementary alphanumeric item.

When using the INTO identifier phrase with variable length records, the amount of data moved to the receiver is equal to the length of the variable length record being read.

identifier-1

Identifier-1 is the receiving field. The current record is moved from the record area to that specified by identifier-1 according to the rules of the MOVE statement without the CORRESPONDING phrase. The following usage notes apply:

- The size of the current record depends on the rules specified in the RECORD clause
- If the file description entry contains a RECORD IS VARYING clause, the move is a group move
- The implied MOVE statement occurs only if the execution of the READ statement is successful
- Subscripting or reference modification associated with identifier-1 applies after reading the record and immediately before it is moved to the data item
- The record is available in both the record area and the data item referenced by identifier-1
- The INTO phrase is allowed in a READ statement only if
  - Only one record description is subordinate to the file description entry, or
  - All record names associated with file-name-1, and the item referenced by identifier-1, describe a group item or an elementary alphanumeric item.
- The record areas associated with file-name-1 and identifier-1 must not be the same storage area

[IBM Extension]

- Identifier-1 can be a floating-point data item.
- Identifier-1 can be a DBCS data-item.
- Identifier-1 can be a date-time data-item.

[End of IBM Extension]

[IBM Extension]

NO LOCK Phrase

The NO LOCK phrase prevents the READ operation from obtaining record locks on files that you open in I-O (update) mode. In addition, a READ operation bearing the NO LOCK phrase will be successful even if the record that is to be made available has been locked by another job. A READ statement bearing this phrase releases records that have been locked by a previous READ operation.

If the DUPLICATES phrase is specified for the file, a record that is read by a statement with the NO LOCK phrase cannot be processed by a DELETE or REWRITE statement.

If you use the NO LOCK phrase for a file that is not open in I-O mode, you will receive an error message at compilation time.

For information about file and record locking, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

FORMAT Phrase

The FORMAT phrase applies only when the READ statement is performed against an indexed file for which the ASSIGN specified DATABASE as the file device type.

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.

Identifier-2, if specified, must be an alphanumeric data item of 10 characters or less.

Literal-1, if specified, must be an uppercase character-string of 10 characters or less.

If the FORMAT phrase is not specified, the first format defined is used when accessing indexed files in random access mode.

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.

When the file is read in **sequential access mode**, 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.

When the file is read in **random access mode**, 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, an INVALID KEY condition is raised. 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 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.

When the file is read in **dynamic access mode**, the next record made available is determined as follows:

Record	FORMAT Phrase	
	Specified	**Omitted**
NEXT	The next record in the keyed sequence access path having the specified format is made available.	The next record in the keyed sequence access path is made available regardless of its format.
PRIOR	The record in the keyed sequence access path preceding the record identified by the file position indicator having the specified format is made available.	The record in the keyed sequence access path preceding the record identified by the file position indicator is made available regardless of its format.
FIRST	The first record in the keyed sequence access path having the specified format is made available.	The first record in the keyed sequence access path is made available regardless of its format.
LAST	The last record in the keyed sequence access path having the specified format is made available.	The last record in the keyed sequence access path is made available regardless of its format.
None of the above	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, an INVALID KEY condition is raised. This occurs even when there are records that have the defined key, but that have a different record format.	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.

NULL-KEY-MAP IS Phrase

The NULL-KEY-MAP IS phrase indicates the value of the identifier, which corresponds to the null-byte map value supplied by data management for the key of the record to be processed. The identifier can be subscripted or reference modified.

The phrase can only be specified for an indexed file for which the ASSIGN clause specified DATABASE as the device type, and the ALWNULL attribute.

If the file has alternate keys, identifier-5 is associated with the null key map of the current key of reference.

For more information about using null-capable fields, refer to the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

NULL-MAP IS Phrase

The NULL-MAP IS phrase indicates the value of the identifier, which corresponds to the null-byte map value supplied by data management for the record to be processed. The identifier can be subscripted or reference modified.

This phrase can be specified for any file for which the ASSIGN clause specified DATABASE as the device type, and the ALWNULL attribute.

For more information about using null-capable fields, refer to the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

[End of IBM Extension]

AT END Phrase

The AT END phrase applies only when a file is read in sequential access mode or dynamic access mode.

If a next record does not exist 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 take place:

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.

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 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 the file.

Note: A sequential read is any READ statement for a file with sequential access, or a READ NEXT, READ PRIOR, READ FIRST, or READ LAST statement for a file with dynamic access.

A READ FIRST or READ LAST statement will result in an AT END condition if the file is empty, or if the FORMAT phrase has been used and no records in the file have the specified format.

When a file is being read in sequential access mode and the AT END condition is recognized, a successful CLOSE statement followed by a successful OPEN statement must be processed for this file before a further READ statement is attempted.

When a file is being read in dynamic access mode and the AT END condition is recognized, one of the following must be processed for the file before a further READ NEXT or READ PRIOR statement is attempted:

- A successful CLOSE statement followed by a successful OPEN statement.
- A successful START statement.
- A successful random access READ statement.
- A successful READ FIRST or READ LAST statement.

If a READ statement for a file with sequential access, or a READ NEXT or READ PRIOR statement for a file with dynamic access, is attempted after the AT END condition has occurred, and the file position indicator has not been reset by one of the specified methods, then a file status of 46 will be returned. Neither the AT END phrase nor the NOT AT END phrase will be executed.

NOT AT END Phrase

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.

INVALID KEY Phrase

The INVALID KEY phrase applies only when a relative or indexed file is read in random access mode or dynamic access mode.

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 250.

NOT INVALID KEY Phrase

The NOT INVALID KEY phrase applies only when a relative or indexed file is read in random access mode or dynamic access mode.

After the successful completion of a READ statement with the NOT INVALID KEY phrase, control transfers to the imperative statement associated with the phrase.

NEXT Phrase

The NEXT phrase applies only for dynamic access mode.

When a relative file is read dynamically and the NEXT phrase is specified, a sequential read is done. When omitted, a random read is done.

When an indexed file is read dynamically and the NEXT phrase is specified, a sequential read is done. If NEXT, FIRST, LAST and PRIOR are all omitted, a random access read is done.

If a READ NEXT operation is performed on a block of records, a READ PRIOR operation cannot occur until the block is empty. If a READ PRIOR operation is performed first, a READ NEXT operation cannot occur until the block is empty. If this is attempted, a file status of 9U will result. To recover from file status 9U, close the file, then open it again.

[IBM Extension]

FIRST Phrase
The FIRST phrase applies only when indexed files are accessed dynamically. If NEXT, FIRST, LAST and PRIOR are all omitted, a random access read is done.

LAST Phrase
The LAST phrase applies only when indexed files are accessed dynamically. If NEXT, FIRST, LAST and PRIOR are all omitted, a random access read is done.

PRIOR Phrase
The PRIOR phrase applies only when indexed files are accessed dynamically. When specified, a sequential read is done. If NEXT, FIRST, LAST and PRIOR are all omitted, a random access read is done.

If a READ PRIOR operation is performed on a block of records, a READ NEXT operation cannot occur until the block is empty. If a READ NEXT operation is performed first, a READ PRIOR operation cannot occur until the block is empty. If this is attempted, a file status of 9U will result. To recover from file status 9U, close the file, then open it again.

[End of IBM Extension]

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 243.

Sequential Files

Sequential files can be read from the following device types:

- TAPEFILE
- DISKETTE
- DISK
- DATABASE

Sequential files can only be read in **sequential access mode**.

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.

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.

- Special Considerations for Device Types TAPEFILE and DISKETTE

Special Considerations for Device Types TAPEFILE and DISKETTE

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.

Relative Files

Relative files can be read from the following device types:

- DISK
- DATABASE

Relative files can be read in **sequential**, **random**, or **dynamic access modes**.

When a relative file is read in **sequential access mode**, 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.

When a relative file is read in **random access mode**, 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.

Indexed Files

Indexed files can be read from the following device types:

- DISK
- DATABASE

Indexed files can be read in **sequential**, **random**, or **dynamic access modes**.

When an indexed file is read in **sequential access mode**, 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]

When an indexed file is read in **random access mode**, the record in the file with a key value equal to that of the current key of reference is then made available. If the file does not contain such a record, the INVALID KEY condition exists, and READ statement processing is unsuccessful. If the FORMAT phrase is not specified on the I-O statement when indexed files are read in random access mode, the first format defined in the file is used. Note that if externally described keys are being used and no format is specified, the first format included in the program is the one used to build the key. This format may not necessarily be the first format in the file.

If the KEY phrase is not specified, the prime RECORD KEY becomes the key of reference for this request. When dynamic access is specified, the prime RECORD KEY is also used as the key of reference for subsequent executions of sequential READ statements, until a different key of reference is established.

When the KEY phrase is specified, data-name-1 becomes the key of reference for this request. When dynamic access is specified, this key of reference is used for subsequent executions of sequential READ statements, until a different key of reference is established.

[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).

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 OPTION parameter, or the DUPKEYCHK option of the PROCESS statement.

[End of IBM Extension]

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 22 on page 373 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.

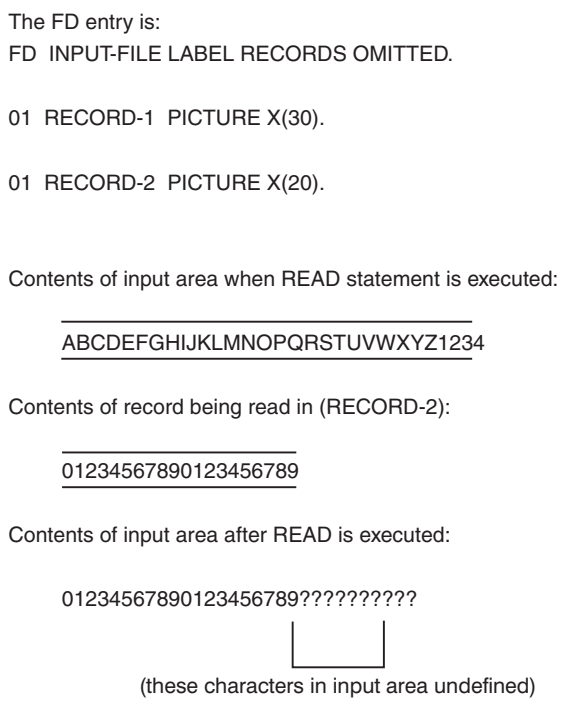


Figure 22. READ Statement with Multiple Record Description

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.

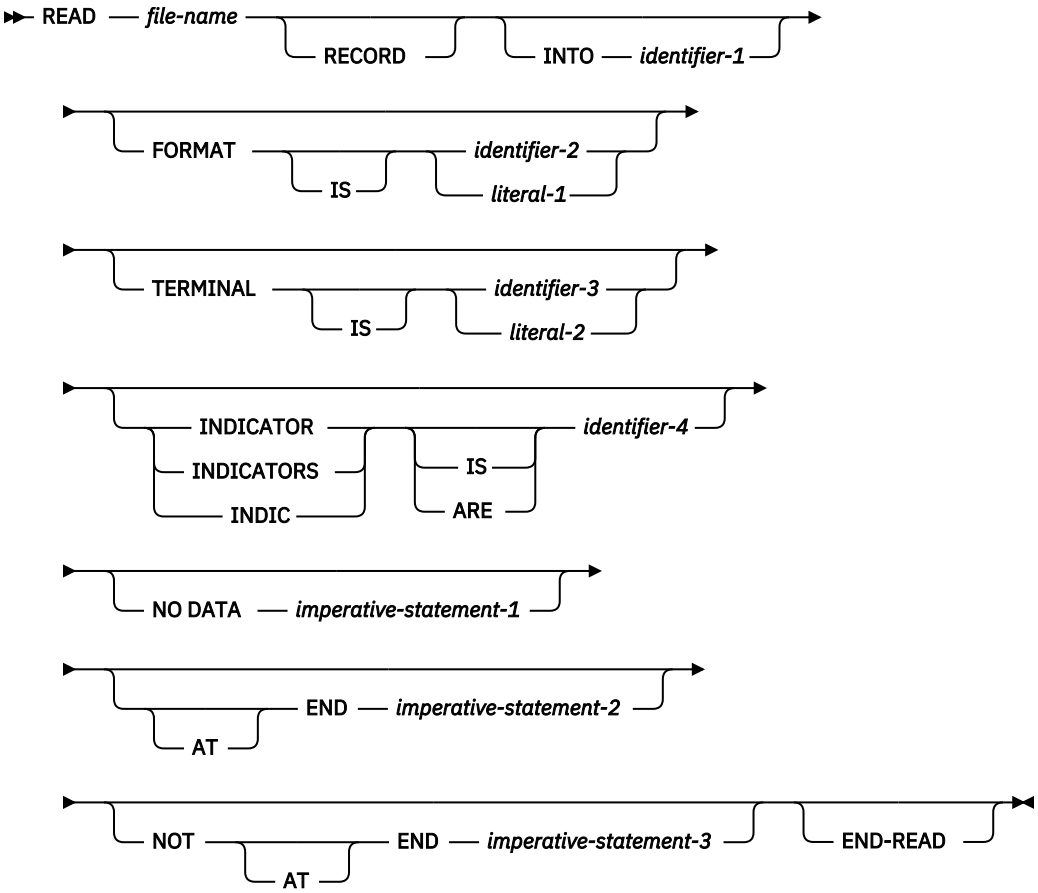
[IBM Extension] Transaction Files

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.

[End of IBM Extension]

READ Statement - Format 4 - Transaction (Nonsubfile)

READ Statement - 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 and 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 whether:

- The READ is for a single device file or a multiple device file
- A specific program device has been requested through the TERMINAL phrase
- A specific record format has been requested through the FORMAT phrase

- 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[3] FORMAT[3] 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 workstation 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 re-invited, 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. For information on how this is determined for work stations, refer to the *File Systems and Management* section of the *Database and File Systems* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/. For communications devices, see the *ICF Programming* manual 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 0A 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 *Application Display Programming* manual for display stations, and the *ICF Programming* manual 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.

[3] 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.

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 workstation devices, refer to the *ICF Programming* book. 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 Programming* book.
   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 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.

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.

INDICATOR Phrase, INDICATORS Phrase, INDIC Phrase
Specifies which indicators are to be read when a data record is read. Indicators can be used to pass information about the data record and how it was entered into the program.

Identifier-4 must be either an elementary Boolean data item specified without the OCCURS clause or a group item that has elementary Boolean data items subordinate to it.

For detailed information on the INDICATORS phrase, refer to Using Indicators with Transaction Files in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

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. The AT END condition occurs when there are no invited program devices and the timer function is not in effect.

The AT END phrase should be specified when 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.

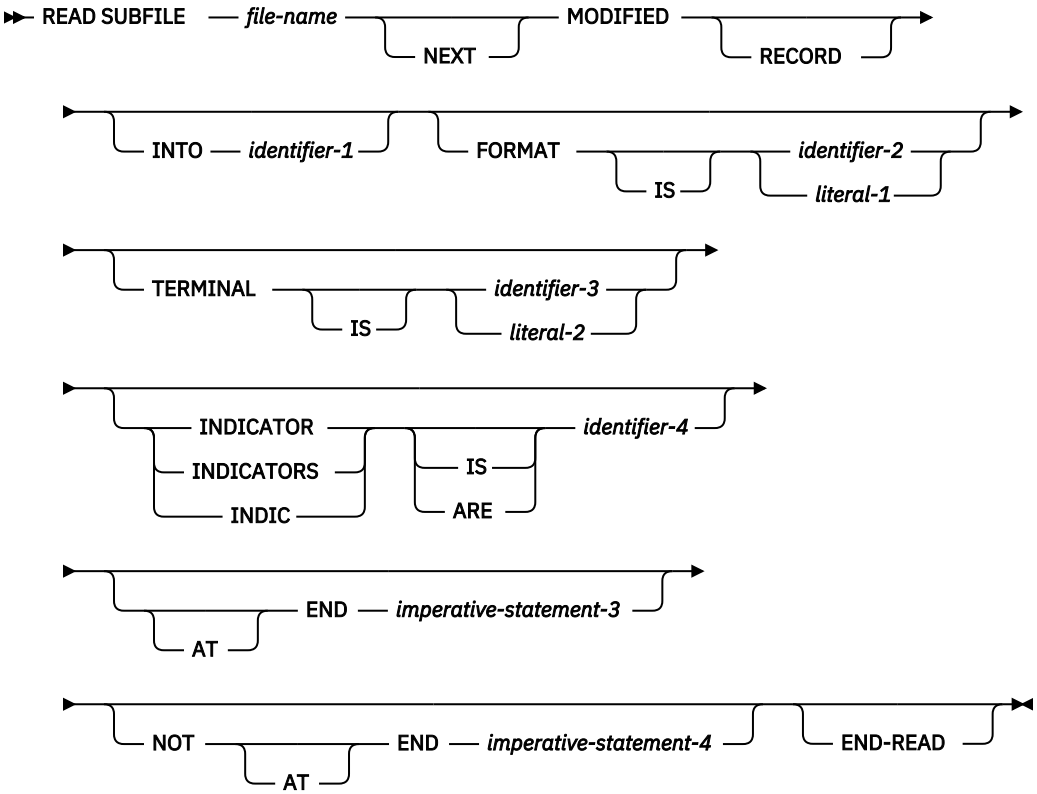
NOT AT END Phrase
This phrase allows you to specify procedures that will be performed when the AT END condition does not exist for the statement that is used.

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 243.

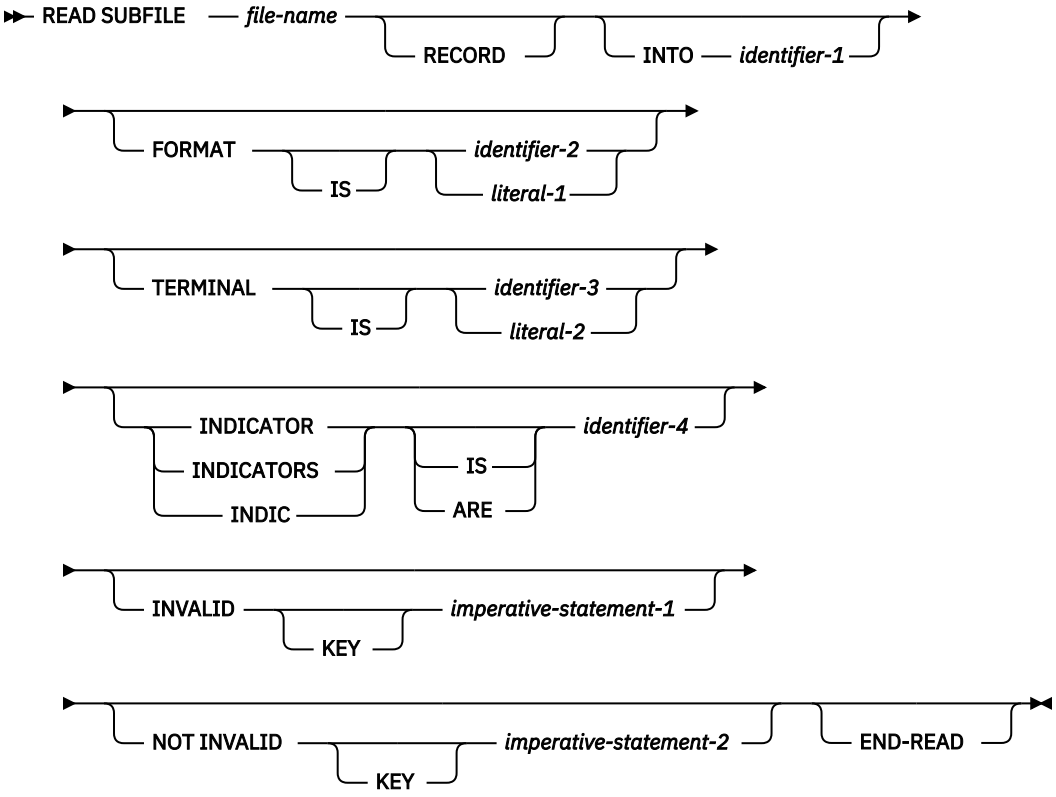
READ Statement - Format 5 - Transaction (Subfile)

READ - Format 5a - Transaction (Subfile Sequential)



Format 5a is used to read a format that is a subfile record, in sequential access mode. 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.

READ - Format 5b - Transaction (Subfile Random)



Format 5b is used to read a format that is a subfile record, in random access mode. The INVALID KEY phrase can only be used for random access of subfile records. The NEXT MODIFIED phrase must not be used to randomly access subfile records.

Format 5a or 5b 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.

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).

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 not updated. Imperative-statement-2, or any applicable USE AFTER ERROR/EXCEPTION procedure, 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 must 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 5a or 5b 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.

INDICATOR Phrase, INDICATORS Phrase, INDIC Phrase

Specifies which indicators are to be read when a data record is read. Indicators can be used to pass information about the data record and how it was entered into the program.

For detailed information on the INDICATORS phrase, refer to Using Indicators with Transaction Files in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* .

Identifier-4 must be either an elementary Boolean data item specified without the OCCURS clause or a group item that has elementary Boolean data items subordinate to it.

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.

The INVALID KEY phrase must 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 250.

NOT INVALID KEY Phrase
This phrase specifies the 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.

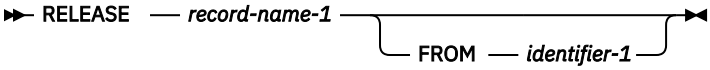
For more information, see "Delimited Scope Statements" on page 243.

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.

RELEASE 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 record in a sort-merge file description entry (SD). Record-name-1 may be qualified.

[IBM Extension] Can be a floating-point or date-time data item. [End of IBM Extension]

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.

Identifier-1 can be the name of an alphanumeric or DBCS function identifier.

[IBM Extension] Identifier-1 must be a DBCS data item if record-name-1 is a DBCS data item. [End of IBM Extension]

[IBM Extension] Identifier-1 can be a floating-point or date-time data item. [End of IBM Extension]

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.

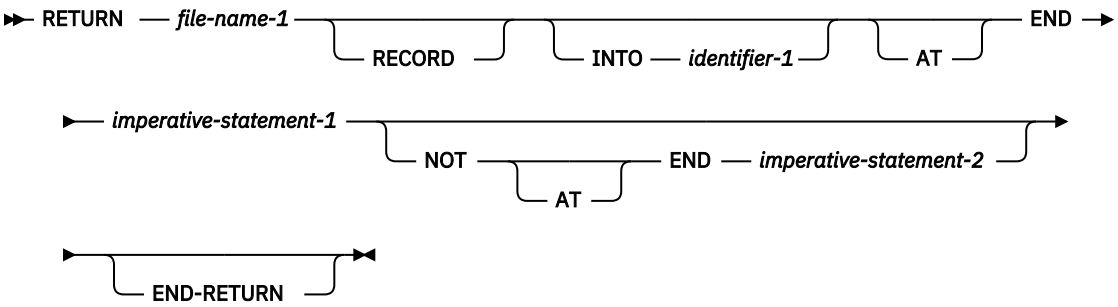
When control passes from the INPUT PROCEDURE, the sort file consists of all those records placed in it by execution of RELEASE statements.

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.

RETURN 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.

The record areas associated with file-name-1 and identifier-1 must not be the same storage area.

The record is available in both the record area and the data-item referenced by identifier-1.

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
```

[IBM Extension] Identifier-1 can be a DBCS, floating-point, or date-time data item. [End of IBM Extension]

Moving takes place according to the rules for the MOVE statement without the CORRESPONDING phrase.

The size of the current record is determined by rules specified for the RECORD clause. If the file description entry contains a RECORD IS VARYING clause, the implied MOVE is a group move. However, the implied MOVE does not occur when the RETURN statement is unsuccessful.

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 or an elementary alphanumeric item.

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 243.

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.

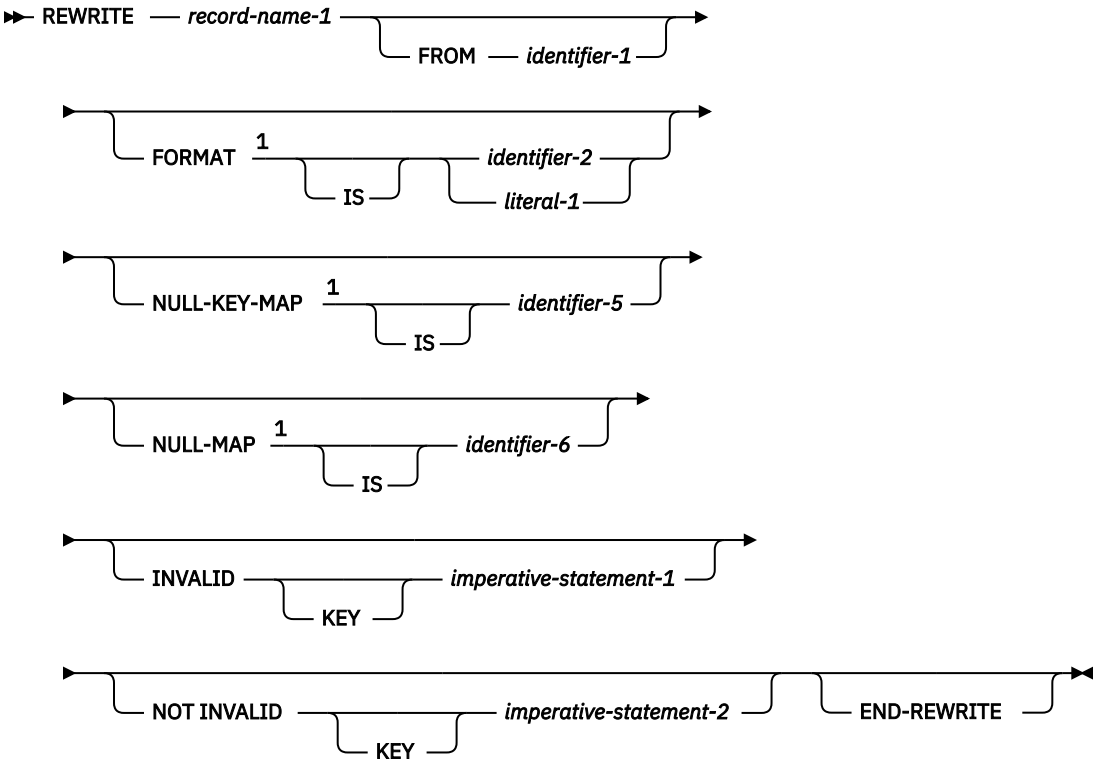
[IBM Extension]

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.

For more information on this command, see the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

[End of IBM Extension]

REWRITE Statement - Format 1



REWRITE Statement - Format 1

Notes:

[1] IBM Extension

record-name-1

The name of a record in the File Section, having the same number of character positions as the record being replaced. The name of the record cannot be reference modified.

identifier-1

This is the sending item.

FROM phrase

This phrase has the following effect:

```
MOVE identifier-1 TO record-name-1.
REWRITE record-name-1.
```

After successful processing of the REWRITE statement, the current record is no longer available in record-name-1, but is still available in identifier-1. Both record-name-1 and identifier-1 cannot refer to the same storage area.

[IBM Extension]

FORMAT phrase

This phrase applies when the REWRITE statement is performed against an indexed file for which the ASSIGN specified DATABASE as the file device type. It is optional when processing a file that has one record format.

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.

Identifier-2, if specified, must be an alphanumeric data item of 10 characters or less.

Literal-1, if specified, must be an uppercase character-string of 10 characters or less.

If the FORMAT phrase is not specified, the first format defined is used when accessing indexed files in Random Access Mode.

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.

NULL-KEY-MAP IS phrase
Refer to the description supplied for this phrase on page NULL-KEY-MAP IS Phrase.

NULL-MAP IS phrase
Refer to the description supplied for this phrase on page NULL-MAP IS Phrase.

[End of IBM Extension]

INVALID KEY phrase
This phrase is valid in indexed organization files, and relative organization files with random or dynamic access. It is processed when the record specified by the key field in the record area is not found.

When an INVALID KEY condition exists, the updating operation does not take place. The data in record-name is unaffected. This phrase transfers control to the corresponding imperative-statement, as appropriate.

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 value of an ALTERNATE RECORD KEY data item for which DUPLICATES is not specified is equal to that of a record already 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 250 for more information.

For sequentially accessed indexed files on device type DISK, this phrase is 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.

NOT INVALID KEY phrase
This phrase is valid for indexed organization files, and relative organization files with random or dynamic access. After the successful completion of a REWRITE statement with the NOT INVALID KEY phrase, control transfers to the imperative statement associated with the phrase.

REWRITE Statement Considerations

After a successful execution of a REWRITE statement, the 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.

Sequential Files

The last input/output statement that was executed for the file must have been a successful READ statement. The record to be replaced is the record that was retrieved by that statement.

The INVALID KEY and NOT INVALID KEY phrases must not be specified. An EXCEPTION/ERROR procedure may be specified.

For files with sequential organization, the number of characters in record-name-1 must equal the number of character positions in the record being replaced.

Indexed Files

If the access mode is sequential, the last input/output statement that was executed for the file must have been a successful READ statement. The record to be replaced is the record that was retrieved by that statement. The value of the RECORD KEY data-item must not have been changed since the record was read. If the value has been changed, then an INVALID KEY condition exists.

If the access mode is random or dynamic, 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, then an INVALID KEY condition exists.

An INVALID KEY phrase should be specified if no EXCEPTION/ERROR procedure has been defined for the file.

For files with indexed organization, the number of characters in record-name-1 can be different from the number of character positions in the record being replaced.

[IBM Extension]

When EXTERNALLY-DESCRIBED-KEY is specified for the file, the key data held in the record area that corresponds to the format specified by the FORMAT phrase (or to the first format, if the FORMAT phrase is not used) is used to determine the current value of the RECORD KEY data-item.

When the WITH DUPLICATES phrase is specified for the file, then for all access modes (sequential, random, and dynamic) the last input/output statement that was executed for the file must have been a successful READ statement. The record to be replaced is the record that was retrieved by that statement. The value of the RECORD KEY data-item must not have been changed since the record was read. If the value has been changed, then an INVALID KEY condition exists.

Note: The READ statement is required to ensure that the correct record is replaced when duplicate keys can be present in the file. The only way to rewrite one specific record from a sequence of records with duplicate keys is to read each of the records sequentially, and rewrite the required record when it has been identified.

[End of IBM Extension]

Relative Files

If the access mode is sequential, the last input/output statement that was executed for the file must have been a successful READ statement. The record to be replaced is the record that was retrieved by that statement. The INVALID KEY and NOT INVALID KEY phrases must not be specified. An EXCEPTION/ERROR procedure may be specified.

If the access mode is random or dynamic, 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, then an INVALID KEY condition exists. An INVALID KEY phrase should be specified if no EXCEPTION/ERROR procedure has been defined for the file.

For files with relative organization, the number of characters in record-name-1 can be different from the number of character positions in the record being replaced.

[IBM Extension] Record Locking

A successful READ statement must precede any REWRITE statement for the following file types:

- Files with sequential organization
- Files using squential access mode
- Files with indexed organization and with duplicate keys.

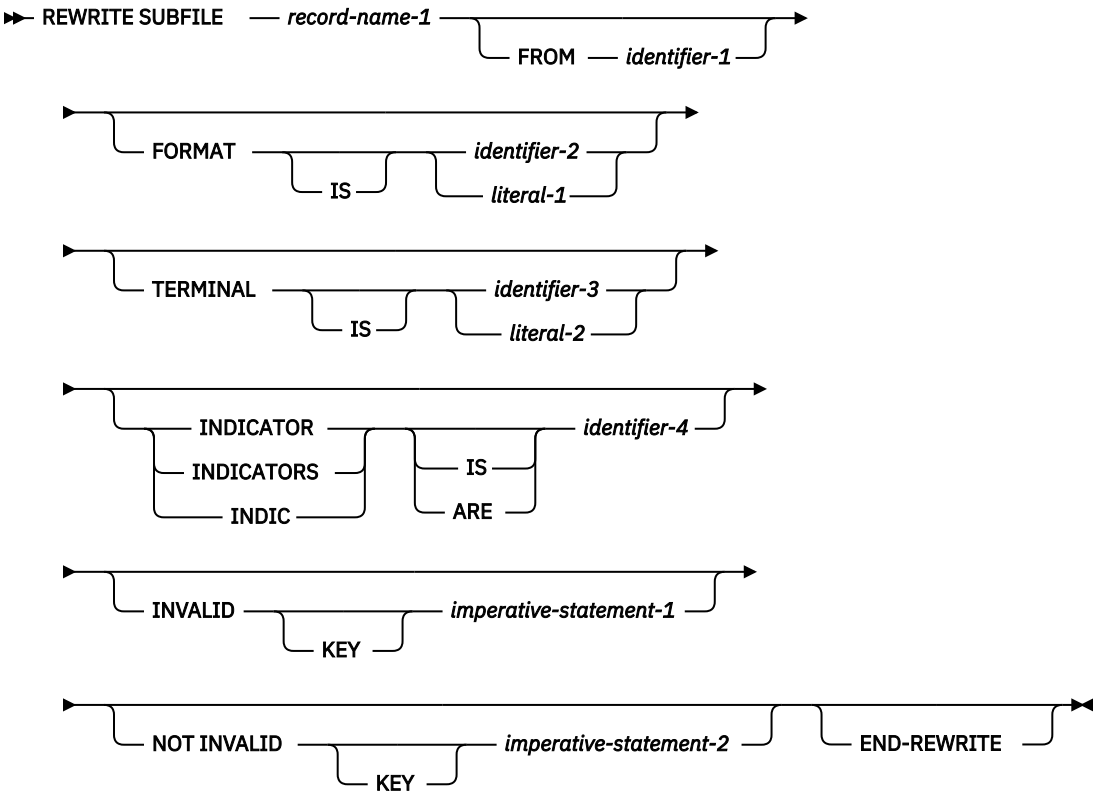
Such a READ statement must not include the NO LOCK phrase. If an attempt is made to replace a record that has been selected by a READ statement, and that record was not locked when it was read, the REWRITE statement will be unsuccessful.

[End of IBM Extension]

[IBM Extension] Transaction (Subfile) Format

The REWRITE statement is used to replace a subfile record that already exists in the subfile.

REWRITE Statement - Format 2 - Transaction (Subfile)



The number of character positions in the record referenced by record-name-1 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.

Note:

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 workstation 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.

INDICATOR Phrase, INDICATORS Phrase, INDIC Phrase
Specifies which indicators are to be used when a data record is rewritten. Indicators can be used to pass information about the data record and how it was entered into the program.

For detailed information on the INDICATORS phrase, refer to Using Indicators with Transaction Files in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* .

Identifier-4 must be either an elementary Boolean data item specified without the OCCURS clause or a group item that has elementary Boolean data items subordinate to it.

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 243.

[End of IBM Extension]

[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.

ROLLBACK Statement - 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. Note that when a file is cleared while being opened for OUTPUT, execution of a ROLLBACK statement does not restore cleared records to the file.

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 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 for files under commitment control are released and the records become available to other jobs. Commitment control can be scoped at the job level or the activation group level. (Commitment control is scoped at the activation group level by default.)

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 its position at the previous commitment boundary. This is important to remember if you are doing sequential processing.
- If no COMMIT statement has been issued since the file was opened, the ROLLBACK statement sets the file position indicator to its position at the OPEN.
- The file position indicator is undefined after a ROLLBACK if the file is closed with uncommitted changes.

If commitment control is scoped at the job level, an implicit ROLLBACK of uncommitted records is automatically done for all files under commitment control at the end of every job. Any uncommitted changes to the database are cancelled.

If commitment control is scoped at the activation group level, an implicit commit occurs when the activation group ends normally. If the activation group ends abnormally, an implicit ROLLBACK occurs.

For more information on commitment control, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

[End of IBM Extension]

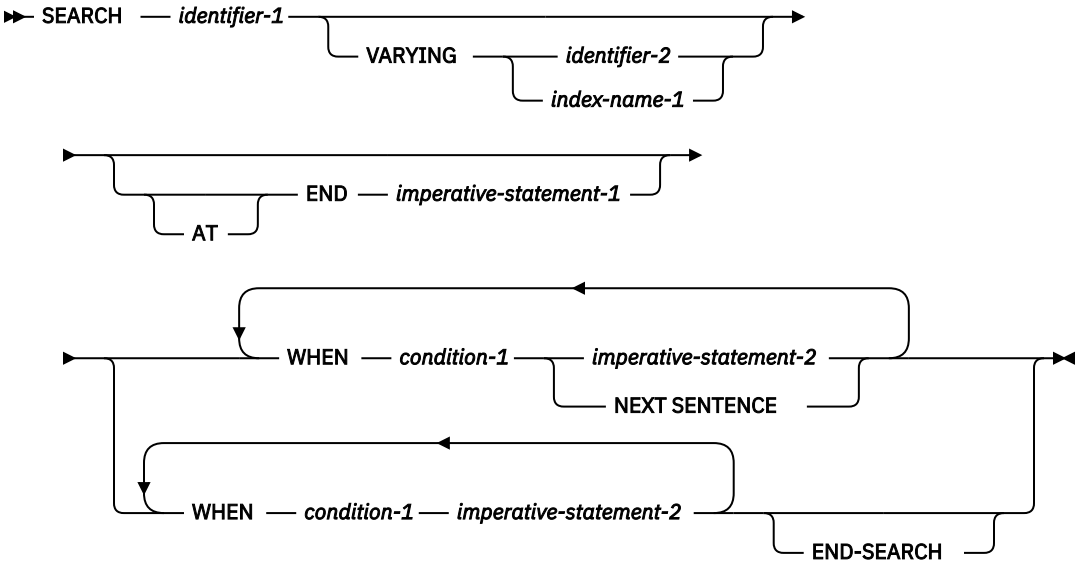
SEARCH Statement

The SEARCH statement searches a table for an element that satisfies the specified condition, and adjusts the associated index to indicate that element.

- SEARCH Statement Considerations
- SEARCH Example

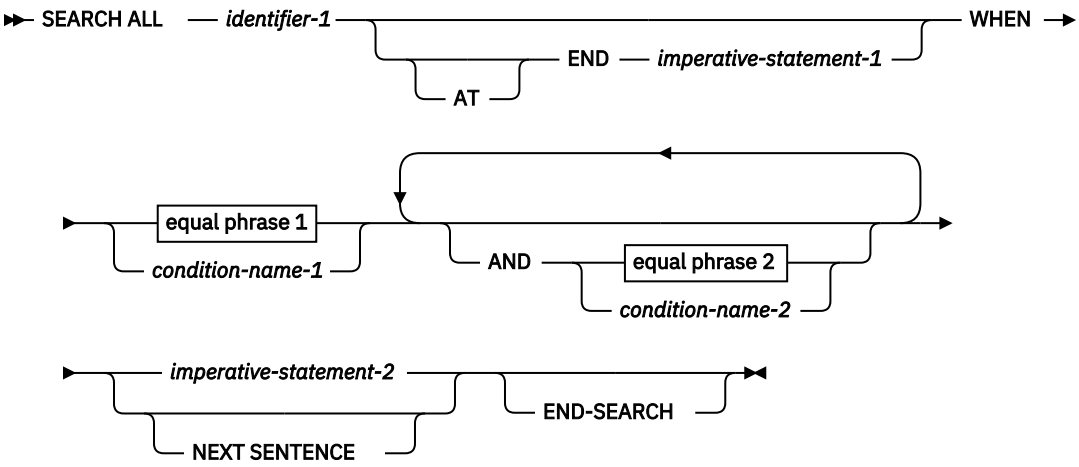
SEARCH Statement - Format 1 - Serial Search

SEARCH Statement - Format 1 - Serial Search

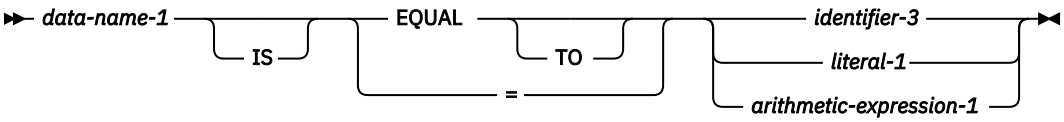


- Execution of a Serial Search

SEARCH Statement - Format 2 - Binary Search

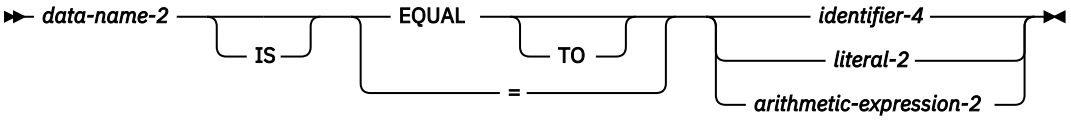


equal phrase 1



SEARCH Statement - Format 2 - Binary Search

equal phrase 2



- Execution of a 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.

[IBM Extension] Identifier-1 can specify a table containing floating-point data items, a table containing DBCS items, or a table containing date-time items. [End of IBM Extension]

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.

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 (see "VARYING Phrase" on page 391). 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.

Condition-1

Condition-1, may be any condition described under "Conditional Expressions" on page 225.

[IBM Extension] Condition-1 can include DBCS relations or DBCS condition-name conditions. [End of IBM Extension]

NEXT SENTENCE Phrase

This phrase causes the transfer of control to an implicit CONTINUE statement immediately preceding the next separator period. If the NEXT SENTENCE phrase is specified, the END-SEARCH phrase must not be specified.

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 243.

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.

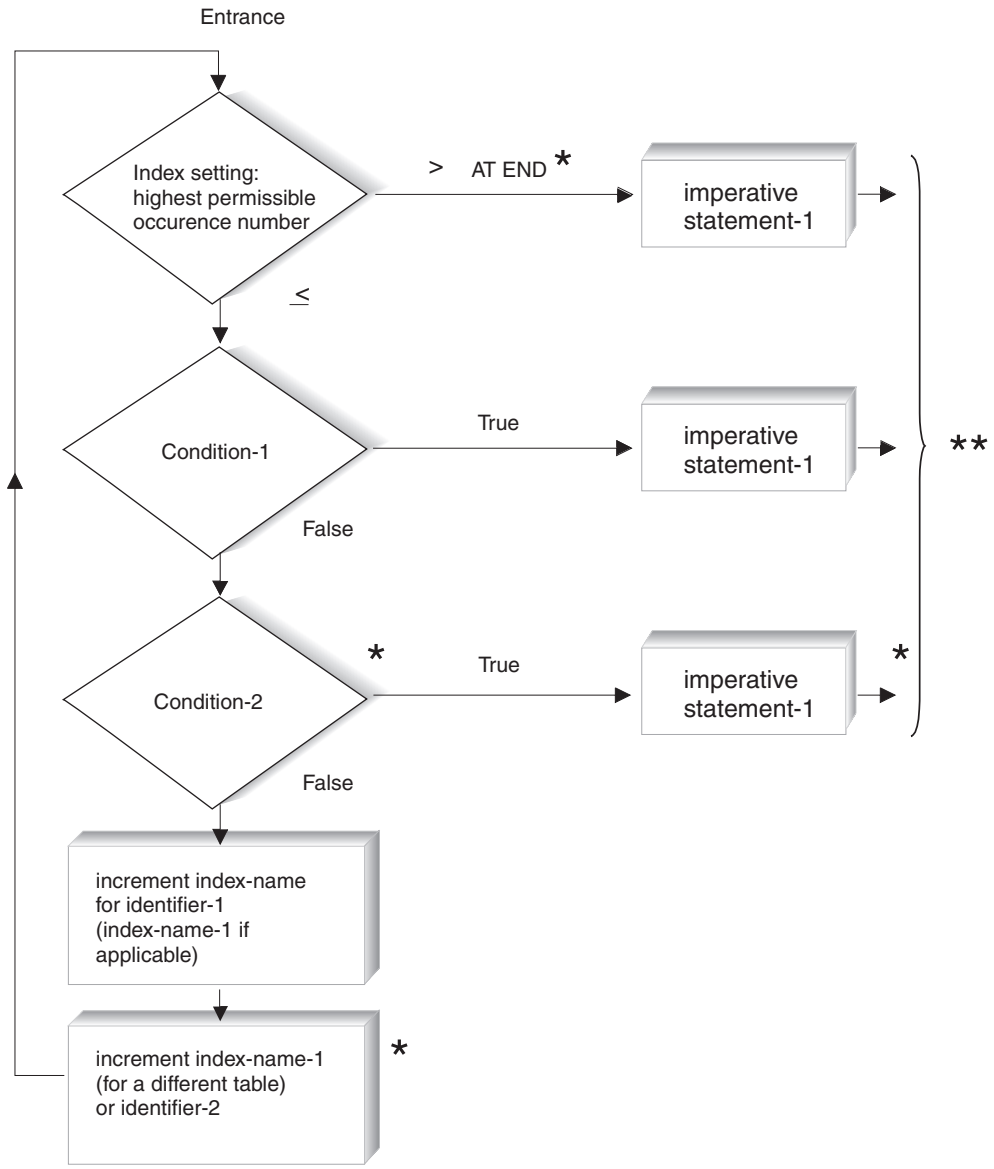
When the VARYING index-name-1 phrase is omitted, the first (or only) index-name for identifier-1 is used for the search.

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.

Figure 23 on page 392 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 23. 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 or a procedure-pointer data item.

[IBM Extension] Identifier-1 cannot be a floating-point data item. [End of IBM Extension]

[IBM Extension] Identifier-1 can be a DBCS data item if the ASCENDING/DESCENDING KEY is defined as a DBCS data item. [End of IBM Extension]

[IBM Extension] Identifier-1 can be a date-time data item if the ASCENDING/DESCENDING KEY is defined as a date-time data item. [End of IBM Extension]

The Data Division description of identifier-1 must contain an OCCURS clause with the INDEXED BY option. For Format-2, the Data Division description must also contain the KEY IS phrase in its OCCURS clause.

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-1, condition-name-2
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-1, data-name-2
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.

[IBM Extension] Data-name-1 or data-name-2 cannot be a floating-point data item. Data-name-1 or data-name-2 can be a date-time data item. [End of IBM Extension]

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 or procedure-pointer data item.

[IBM Extension] Can be floating-point or date-time data items. [End of IBM Extension]

arithmetic-expression-1, arithmetic-expression-2
May be any of the expressions defined under “Arithmetic Expressions” on page 223, 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 can be used to:

- Initialize the values of index-names or identifiers used to reference table elements
- Increment or decrement an index-name
- Set the status of an external switch to ON or OFF
- Move data to make conditional variable conditions true

[IBM Extension]

- Set pointer and procedure-pointer data items and the ADDRESS OF special register
- Set and query the locale categories of the current locale.

[End of IBM Extension]

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.

[IBM Extension]

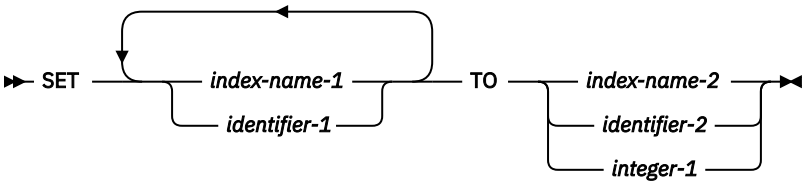
- Format 5 - Pointer Data Item
- Format 6 - Procedure-pointer Data Item
- Format 7 - Adjusting Pointers
- Format 8 - Locale

[End of IBM Extension]

Format 1 - Initializing Index-names, Identifiers

When Format 1 of the SET statement is executed, the current value of the receiving field is replaced by the value of the sending field (with conversion).

SET Statement - Format 1



index-name-1, identifier-1
Receiving fields.

Must be either index data items or elementary numeric integer items.

[IBM Extension] Identifier-1 cannot be a floating-point data item. [End of IBM Extension]

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 be either an index data item or an elementary numeric integer item.

[IBM Extension] Identifier-2 cannot be a floating-point data item. [End of IBM Extension]

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, and so on) are usually 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, 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, 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 34 on page 396 shows valid combinations of sending and receiving fields in a Format 1 SET statement.

Table 34. 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*	—

Table 34. Sending and Receiving Fields for Format 1 SET Statement (continued)			
Sending Field	**Receiving Field**		
	Index-name	**Index Data Item**	**Integer Data Item**
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.

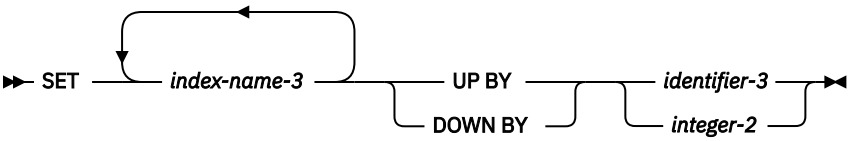
The value for an index-name after execution of a SEARCH or PERFORM statement may be undefined; therefore, a Format 1 SET statement should reinitialize such index-names before other table-handling operations are attempted.

[IBM Extension] If index-name-2 refers to a table that has a subordinate item that contains an OCCURS DEPENDING ON clause, identifier-1 may receive undefined values. For more information, see “Appendix H. Complex OCCURS DEPENDING ON” on page 585. [End of IBM Extension]

Format 2 - Adjusting Index Values

When Format 2 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.

SET Statement - Format 2



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.

identifier-3
This sending field must be an elementary integer data-item.

[IBM Extension] Identifier-3 cannot be a floating-point data item. [End of IBM Extension]

integer-2
This sending field must be an integer.

When Format 2 of the 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.

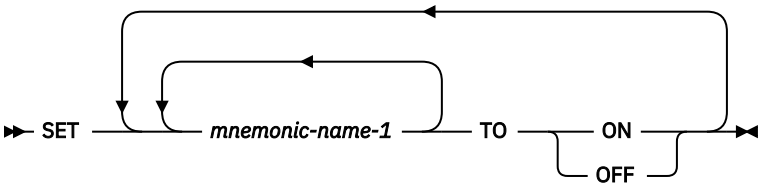
[IBM Extension] If index-name-3 refers to a table that has a subordinate item that contains an OCCURS DEPENDING ON clause, and if the ODO object is changed before executing a Format 2 SET Statement, index-name-3 cannot contain a value that corresponds to an occurrence number of its associated table. For more information, see “Appendix H. Complex OCCURS DEPENDING ON” on page 585. [End of IBM Extension]

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 - Setting External Switches

When Format 3 of the SET statement is executed, the status of each external switch associated with the specified mnemonic-name is turned ON or OFF.

SET Statement - Format 3



mnemonic-name
Must be associated with an external switch, the status of which can be altered.

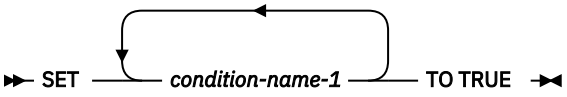
For Format 3 each mnemonic-name must be associated with an external switch, the status of which can be altered. The only external switches allowed are the UPSI switches, UPSI-0 through UPSI-7.

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 - Condition-names

When Format 4 of the SET statement is executed, the value associated with a condition-name is placed in its conditional variable.

SET Statement - Format 4



condition-name-1
Must be associated with a conditional variable.

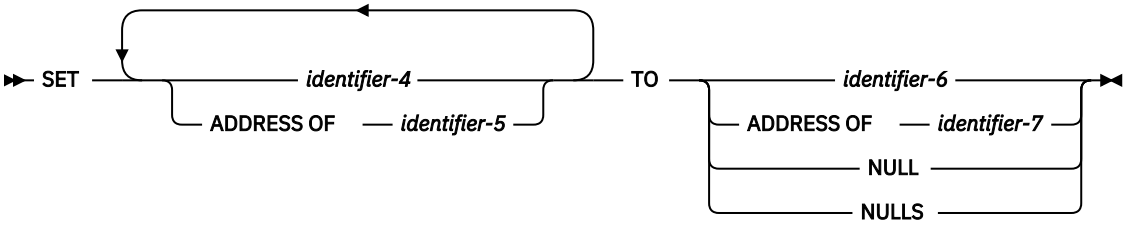
If more than one literal is specified in the VALUE clause of condition-name-1, its associated conditional variable is set to the first literal.

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 the condition-names are specified.

[IBM Extension] Format 5 - Pointer Data Item

When Format 5 of the SET statement is executed, the current value of the receiving field is replaced by the address value contained in the sending field.

SET Statement - Format 5



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, Local-Storage or File sections.

ADDRESS OF identifier-7
Sending field.

Must be an item in the data division 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.

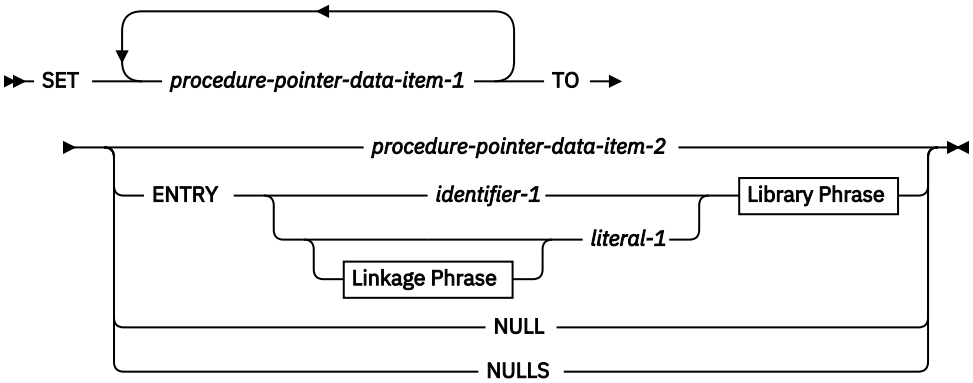
NULL, NULLS
Sending field.

Sets the receiving field to contain the value of an invalid address.

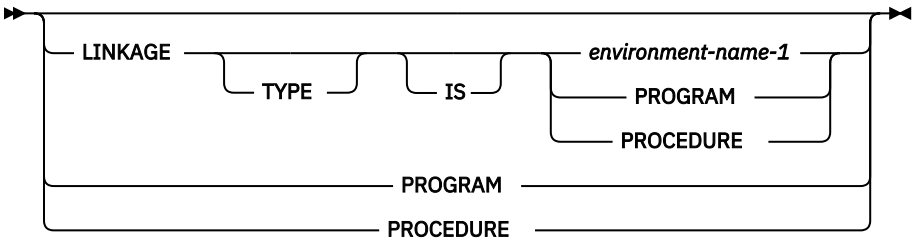
[End of IBM Extension]

[IBM Extension] Format 6 - Procedure-Pointer Data Item

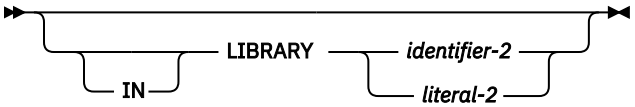
When Format 6 of the SET statement is executed, the current value of the receiving field is replaced by the address value contained in the sending field.



Linkage Phrase



SET Statement - Format 6
Library Phrase



Procedure-pointer-data-item-1, procedure-pointer-data-item-2
Procedure-pointer-data-item-1 is the receiving field.

They must be described as USAGE IS PROCEDURE-POINTER.

identifier-1
Must be defined as an alphanumeric item such that the value can be a program name. (For more information, see "PROGRAM-ID Paragraph" on page 70.) The procedure-pointer data item is set to the outermost COBOL program (an ILE procedure) of the same compilation unit, or to the program object (*PGM), named in identifier-1. The contents of the identifiers are affected by the *MONOPRC option of the CRTCBLMOD or CRTBNDCBL command.

literal-1
Must be nonnumeric and must conform to the rules for formation of program-names. The literals are affected by the *MONOPRC option of the CRTCBLMOD or CRTBNDCBL command. The procedure-pointer data item can be set to the outermost COBOL program (an ILE procedure) of the same compilation unit, the outermost COBOL program (an ILE procedure) in another compilation unit, an ILE procedure (written in another ILE language), or a program object (*PGM). The procedure-pointer data item cannot be set to a nested COBOL program even if the nested COBOL program of the specified name is visible from the point of SET. The LINKAGE TYPE phrase of the ENTRY clause, along with the LINKAGE TYPE clause of the SPECIAL-NAMES paragraph and the LINKLIT parameter of the CRTCBLMOD or CRTBNDCBL command determine the type of object that the procedure-pointer data item is set to.

[End of IBM Extension]

LINKAGE TYPE Phrase

The LINKAGE TYPE phrase is used to specify the type of program that the procedure-pointer data item is set to. It could be set to the address of a separately compiled program object (*PGM) or a procedure within a program.

environment-name-1
The type of program that procedure-pointer-data-item-1 will be set to. Environment-name-1 can be defined as:

- PGM (a program object, or *PGM)
- PRC (a procedure)

PROGRAM
Procedure-pointer-data-item-1 is set to a program object (*PGM).

PROCEDURE
Procedure-pointer-data-item-1 is set to a procedure.

NULL(S)
Sets the receiving field to contain the value of an invalid address.

IN LIBRARY Phrase

The IN LIBRARY phrase is valid only for setting a procedure pointer data item to an IBM i program object. That is, a linkage of program must be specified, whether implicitly or explicitly, on the SET statement.

identifier-2
Must be an alphanumeric data item. The contents of identifier-2 must represent a valid IBM i library name. IBM i library names are at most 10 characters long; the first 10 characters of identifier-2 are used to form the library name.

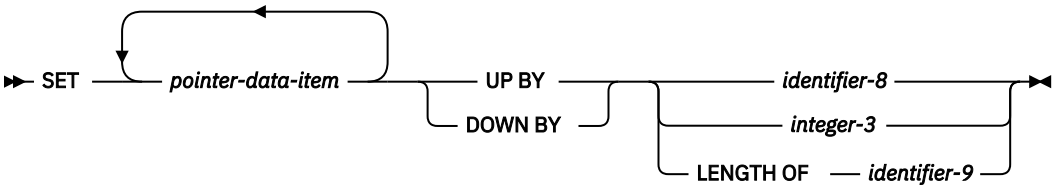
literal-2
Must be nonnumeric and can be a maximum of 10 characters.

Identifier-2 and literal-2 are *not* affected by the *MONOPRC compiler option, and can contain an IBM i extended name.

[IBM Extension] Format 7 - Adjusting Pointers

When Format 7 of the SET statement is executed, the address contained in pointer-data-item is increased (UP BY) or decreased (DOWN BY) by a value that corresponds to the value in the sending field.

SET Statement - Format 7



pointer-data-item
The receiving field must be an elementary data item with USAGE IS POINTER.

identifier-8
This sending field must be an elementary integer data-item.

Identifier-8 cannot be a floating-point data item.

integer-3
This sending field must be an integer.

identifier-9
For more information on the rules for identifier-9, see “LENGTH OF Special Register” on page 286.

[End of IBM Extension]

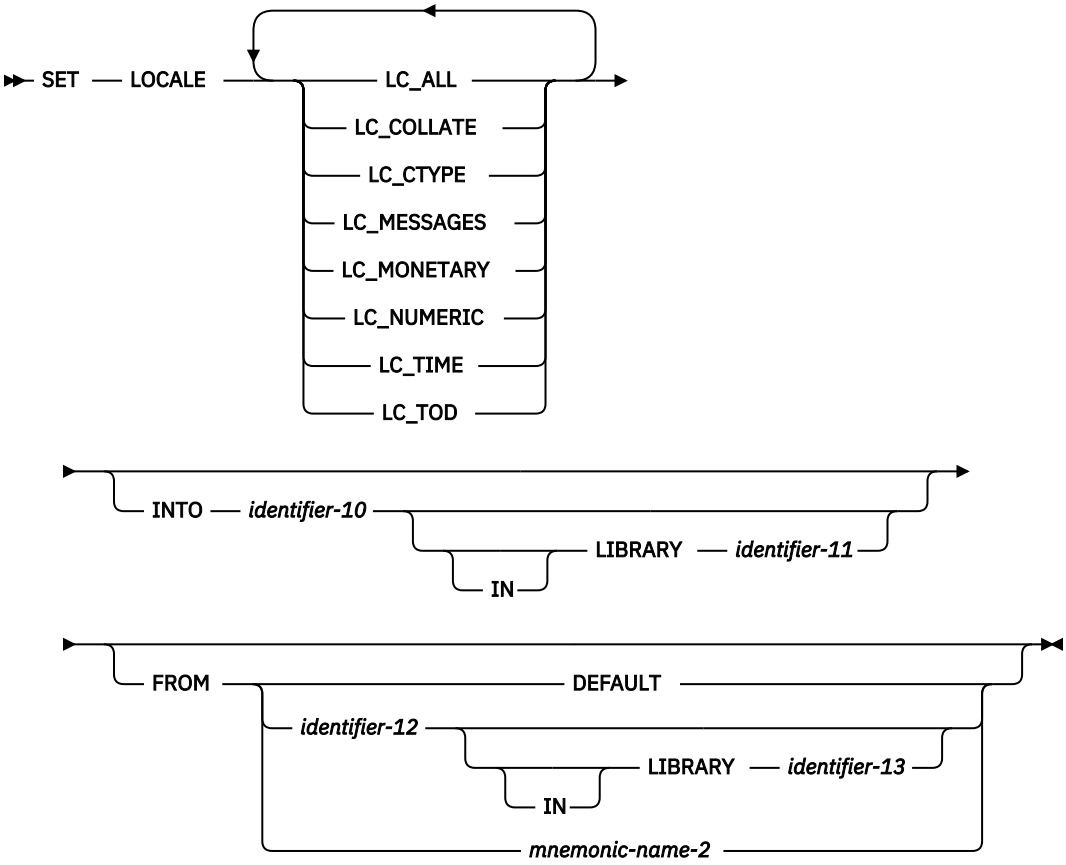
[IBM Extension] Format 8 - Locale

Format 8 of the SET statement allows you to set and query the locale categories of the current locale. A **locale** is a system object containing language and cultural information. For example, a locale contains the appropriate format for a date or time in a particular region of the world. The information in a locale is divided into **locale categories**. For example, locale category LC_TIME contains information about date and time formats. For each run unit there is a DEFAULT locale, a current locale, and from zero to many specific locales. The current locale is altered by setting some or all of its locale categories to the DEFAULT or a specific locale. The name of the specific locale to which a locale category (of the current locale) was set can be placed into an identifier. The contents of a locale category can be changed by setting the locale category from:

- The system default
- A locale defined in an alphanumeric elementary data item
- The mnemonic-name specified in the SPECIAL-NAMES paragraph.

Each locale category specified remains in effect for the duration of the run unit or until another SET statement specifying the category is processed.

Set Statement - Format 8



LC_ALL
Locale categories LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, LC_TIME, and LC_TOD, as well as any other categories included in the locale.

LC_COLLATE
The locale category that defines collation sequence.

LC_CTYPE
The locale category that defines character classification and character type.

LC_MESSAGES
The locale category that defines formatting of informative and diagnostic messages, and interactive responses.

LC_MONETARY
The locale category that defines monetary formatting.

LC_NUMERIC
The locale category that defines numeric formatting.

LC_TIME
Thelocale category that defines date and time formatting.

LC_TOD
The locale category that defines definitions of time zone differences, time zone names, and Daylight Saving Time start and end points.

identifier-10
The value of identifier-10 references a locale-category. Identifier-10 must be an elementary alphanumeric data item. If the INTO phrase is specified, the identification of the current locale for the specified category is stored in the data item referenced by identifier-10. The INTO phrase is processed before the FROM phrase, using the rules of the MOVE statement for an alphanumeric-to-alphanumeric move.

DEFAULT
Sets the locale category to the current default. The default locale exists at the time a run unit is activated, and remains the default for the duration of the run unit. The default locale also becomes the current locale at the time a run unit is activated, and remains the current locale until it is switched using Format 8 of the SET statement.

identifier-12
The value of identifier-10 references a locale-category. Identifier-12 must reference an elementary alphanumeric data item. If the locale specified in identifier-12 is not available, an operating system escape message is issued. If the FROM phrase is specified, the current locale for the specified category is set to the content of the data item referenced by identifier-12. The identification of the current locale is stored using the rules of the MOVE statement for an alphanumeric-to-alphanumeric move.

mnemonic-name-2
If the locale specified in mnemonic-name-2 is not available, an operating system escape message is issued. If the FROM phrase is specified, the current locale for the specified category is set to the locale category identified by mnemonic-name-2.

[End of IBM Extension]

IN LIBRARY Phrase

The IN LIBRARY phrase is used to specify the IBM i library where the locale object exists. For the INTO clause, identifier-11 is updated with the library name for the specified locale category. For the FROM clause, identifier-12 is used to locate the locale object that the locale category will be set to.

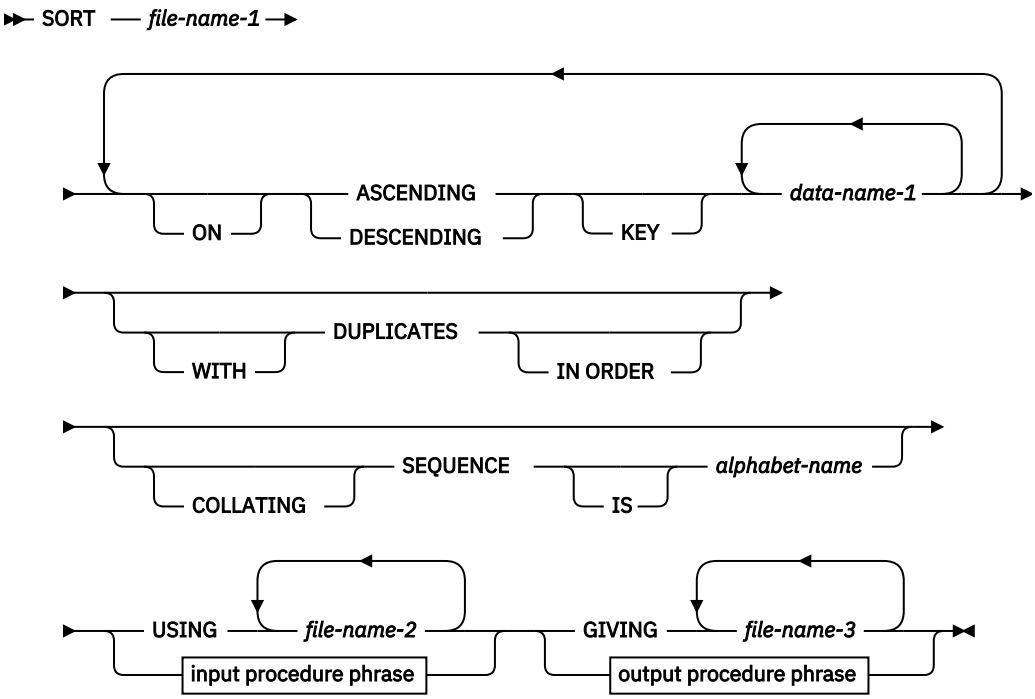
identifier-11, identifier-13
Must be an elementary alphanumeric data item. The contents of identifier-11 or identifier-13 must represent a valid IBM i library name. IBM i library names are at most 10 characters long; the first 10 characters of identifier-2 are used to form the library name.

If identifier-13 is not specified, a library of *LIBL is assumed. Otherwise, identifier-13 must contain the library where the locale object name, specified in identifier-12, exists. If identifier-11 is specified, it will contain the library name of the locale object to which the current locale category was last set. If the locale name in identifier-10 is DEFAULT, identifier-11 will be set to spaces.

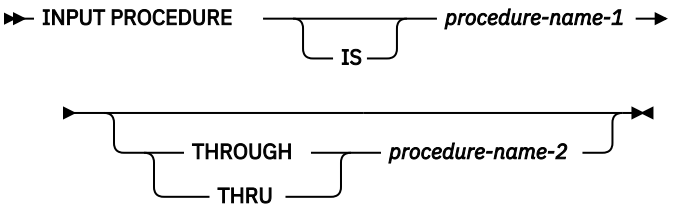
Identifier-11 and identifier-13 are *not* affected by the *MONOPRC compiler option, and can contain an IBM i extended name.

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. A SORT statement may appear anywhere in the Procedure Division except in a Declarative Section. The maximum number of USING or GIVING files is 32.

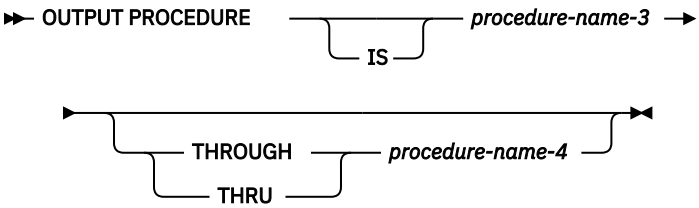


input procedure phrase



SORT Statement - Format

output procedure phrase



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 only supported for DATABASE files that have ALWNULL specified on their ASSIGN clause. If ALWNULL is not specified, the SORT operation will fail, and file status of 90 will be returned if a field contains a null value.

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:

- 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, then the KEY data items need be described in only one of the record descriptions.
- If file-name-1 contains variable length records, all of the KEY data-items must be contained within the first *n* character positions of the record, where *n* equals the maximum record size specified for file-name-1.
- KEY data items must not contain an OCCURS clause or be subordinate to an item that contains an OCCURS clause.
- The total length of the KEY data item must not exceed 2 000 bytes.
- KEY data items can be qualified, but they cannot be subscripted or indexed.
- KEY data items cannot be variably-located.
- Variable length fields can not be used in a SORT key as a variable length field. Variable length fields are converted into group items by ILE COBOL. Since variable length fields are converted into group items, they are compared as alphanumeric data items when used in a SORT key.

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 keywords 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 406).
- If the KEY data item is DBCS, DBCS-edited, or national, the sequence of key values is based on a binary collating sequence of the hexadecimal values of the DBCS or national characters. The COLLATING SEQUENCE phrase is ignored.

[IBM Extension]

- KEY data items can be floating-point or date-time items.
- KEY data items can be reference modified, but they cannot be subscripted or indexed.
- If the KEY is an external floating-point item, the compiler treats the data item as character data, rather than numeric data. The sequence in which the records are sorted depends on the collating sequence used.
- If the KEY data item is internal floating-point, the sequence of key values is in numeric order.
- If the KEY is a date-time item, only some formats will be sorted as date or time items. ILE COBOL supports many more date-time formats than IBM i DDS. In general, ILE COBOL date-time formats that match an IBM i DDS format are sorted as a date or time item; all other formats are treated as alphanumeric items, and are sorted based on their hexadecimal value.

[End of IBM Extension]

- 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 225).

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 78 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.

If the output files contain variable length records, the size of the records contained in file-name-1 must not be less than the smallest record nor greater than the largest described for the output files. If the output files contain fixed length records, the size of the records contained in file-name-1 must not be greater than the largest record described for the output files.

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-3 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 SORT 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.

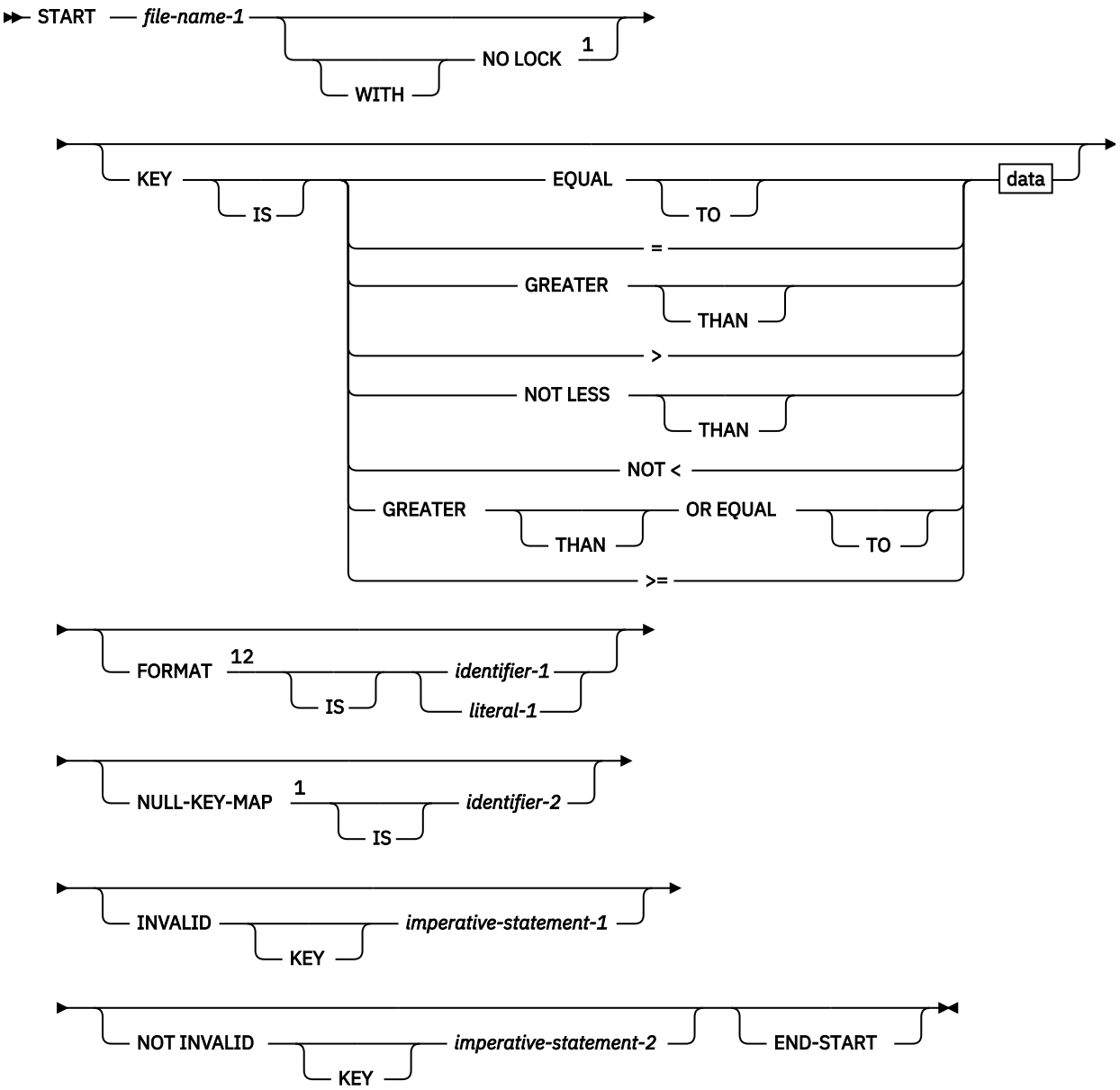
As with the PERFORM statement, execution of the procedure ends after the last statement executes. The last statement in an input or output procedure can be the EXIT statement (see “EXIT Statement” on page 318).

[IBM Extension] The SORT-RETURN special register contains a return code indicating the success (or lack of) of a SORT operation. See “SORT-RETURN Special Register” on page 338 for more information. [End of IBM Extension]

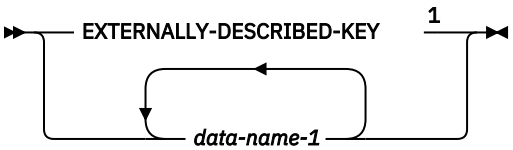
START Statement

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.



data



START Statement - Format

Notes:

[1] IBM Extension

[2] Applies only to indexed files on DATABASE devices

NO LOCK Phrase

[IBM Extension]

The NO LOCK phrase prevents the START operation from obtaining record locks on files that are opened in I-O (update) mode. In addition, a START statement bearing the NO LOCK phrase will be successful

even if the record that satisfies the key value comparison has been locked by another job. A START statement bearing this phrase releases records that have been locked by a previous START operation.

If this phrase is used for a file that is not open in I-O mode, an error message is issued.

For information about file and record locking, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

[End of IBM Extension]

file-name-1
Must be 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] Data-name-1 can be an internal or external floating-point, DBCS, or date-time data item. [End of IBM Extension]

[IBM Extension] Multiple data-names can 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 250.)

[IBM Extension] FORMAT Phrase

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.

Identifier-1, if specified, must be an alphanumeric data item of 10 characters or less.

Literal-1, if specified, must be an uppercase character-string of 10 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.

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 Table 35 on page 413 for a description of how the FORMAT phrase interacts with the EXTERNALLY-DESCRIBED-KEY and KEY IS phrases.

[End of IBM Extension]

[IBM Extension] NULL-KEY-MAP IS Phrase

Specifies the null key map of the record for the START operation according to the value specified for identifier-2. Identifier-2 must be a boolean or alphanumeric item.

Identifier-2 can be subscripted or reference modified.

If the file has alternate keys, identifier-2 is associated with the null key map of the current key of reference.

This phrase can only be specified for a file with the ALWNULL attribute and a device type of DATABASE specified in the ASSIGN clause. If one of the key fields is null-capable and the NULL-KEY-MAP phrase is not used, a null-key-map with all boolean zeroes is used.

[End of IBM Extension]

Example of NULL-KEY-MAP IS Phrase

In this example, the following values represent the key in a file, which contains 3 fields of 2 bytes each. The key is defined by the following code in the File Section:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILE-1 ASSIGN to DATABASE-FILE1-ALWNULL
    ACCESS is DYNAMIC RECORD KEY IS FULL-PRODUCT-CODE IN FILE-1
    ORGANIZATION IS INDEXED.
FD FILE-1.
01 FULL-PRODUCT-CODE.
   05 TYPE-CODE      PIC X(2).
   05 COLOR-CODE     PIC X(2).
   05 LOCATION-CODE  PIC X(2).
WORKING-STORAGE SECTION.
01 FILE1-N.
   05 FULL-PRODUCT-CODE-NKM.
      06 FILLER             PIC X VALUE ZERO.
      06 COLOR-CODE-NF      PIC 1 VALUE B"0".
      06 LOCATION-CODE-NF   PIC 1 VALUE B"0".
```

Fields 2 and 3 are null-capable, where '-' indicates null, and xx indicates any value. The following are representations of the records in the file:

```
NN----
NN--xx
NNxx--
```

Consider the following START statement:

```
  START FILE-1
      NULL-KEY-MAP IS FULL-PRODUCT-CODE-NKM
      INVALID KEY DISPLAY "No data in system for product code " TYPE-CODE
      GO TO ERROR-ROUTINE
      END-START.
```

If the null-key-map in the START statement has a value of `010`, the pointer is set to point to the record with the key `NN--xx`. If the null-key-map in the START statement has a value of `011`, the pointer is set to point to the record with the key `NN----`.

For more information about using null-capable fields, refer to the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

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 250.)

The INVALID KEY phrase must be specified if no EXCEPTION/ERROR procedure is explicitly or implicitly specified for this file.

NOT INVALID KEY Phrase

After successful completion of a START statement with the NOT INVALID KEY phrase, control transfers to the imperative statement associated with the phrase.

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 243.

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 file position indicator is set to the record with a key equal to the value contained in the RECORD KEY data item.

When START statement execution is successful, the RECORD KEY or ALTERNATE RECORD KEY with which data-name-1 is associated becomes the key of reference for subsequent READ statements.

When the KEY phrase is specified, the search argument used for the comparison is data-name-1, which can be:

- The prime RECORD KEY itself
- Any ALTERNATE RECORD KEY
- 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, COMP-4, or COMP-5, 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 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 single 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, COMP-4, or COMP-5 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.
- Table 35 on page 413 shows the action between the KEY IS phrase and the FORMAT phrase:

Table 35. Relationship between KEY IS and FORMAT Phrases

FORMAT Phrase specified	KEY Phrase		
	Data-Name Series	Omitted	EXTERNALLY-DESCRIBED-KEY
Yes	A, B	C, D	C, B
No	A, E	F, G	F, E

A
The search argument is built using the specified data items.

B
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.

C
The search argument is built using the key fields in the record area for the format specified in the FORMAT phrase.

D
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.

E
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.

F
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.

G
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]

When the KEY phrase is not specified, the key data item used for the EQUAL TO comparison is the prime RECORD KEY.

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.

[IBM Extension]

For indexed files of device type DATABASE, 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]

Relative Files

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.

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 relative key and the relative record numbers of existing records in the file.

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 250.)

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.

STOP Statement

The STOP statement halts execution of the object program either permanently or temporarily.

STOP Statement - Format

literal

May be numeric, nonnumeric or Boolean, and may be any figurative constant except ALL literal. If the literal is numeric, it must be an unsigned integer.

[IBM Extension] Cannot be a floating-point literal. [End of IBM Extension]

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 the action to be taken.

Operator
Response Action

G (default)
Continue at next instruction.

C
Terminate the execution of all programs up to and including the program at the nearest control boundary. If the nearest control boundary is a hard control boundary then escape message CEE9901 is issued to the caller of the COBOL run unit. For batch jobs, the job is canceled if the ENDSEV parameter (see CRTJOBD CL command) for the job contains a value that is less than or equal to the severity of the message.

D
Dump COBOL identifiers and then perform the same action as C.

F
Dump COBOL identifiers and file information and then perform the same action as C.

The output of the STOP literal contains the program-name. The literal is contained in the second level text, and is displayed when the Help key is used.

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 all programs up to and including the program at the nearest control boundary is ended, and control is returned to the program prior to the control boundary. If the nearest control boundary is a hard control boundary, then STOP RUN causes the activation group (run unit) to end, which in turn causes all files scoped to the activation group to be closed. If a STOP RUN statement appears in a sequence of imperative statements, it must be the last or only statement in the sequence.

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.

For details on the behavior of the STOP RUN statement under various conditions, see "Returning from an ILE COBOL Program" in *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

[IBM Extension] RETURN-CODE Special Register

The RETURN-CODE special register can be used to pass return code information (that is, a numeric value) from a program to its caller (either a calling program or the system).

You can set the RETURN-CODE special register before executing an EXIT PROGRAM, GOBACK, or STOP RUN statement.

RETURN-CODE has the implicit definition:

```
01 RETURN-CODE GLOBAL PICTURE S9999 USAGE BINARY VALUE 0
```

This special register may be used anywhere in a program where a data-item with a data definition of PICTURE S9999 USAGE BINARY is allowed. When used in nested programs, the RETURN-CODE special register is implicitly defined as GLOBAL in the outermost program. When a COBOL subprogram terminates, the contents of the RETURN-CODE special register of the subprogram are transferred into the RETURN-CODE special register of the calling program. When the main COBOL program terminates, and

control returns to the operating system, the special register content is returned to the operating system as a user return code.

Note that the main COBOL program must be the first program in an activation group, so normally this COBOL program should not be compiled with option ACTGRP(*CALLER), if you want the contents of the RETURN-CODE special register to be returned as a user return code for the job. The user return code can be retrieved by the calling program by calling API QUSRJOBI with format JOBI0600.

For the first call to a program, the RETURN-CODE special register is initialized to zero, which is the normal return code for successful completion. The field will be re-set to zero on subsequent calls to a program that has been cancelled or which possesses the INITIAL attribute. Otherwise, the RETURN-CODE special register will not be re-set, it will be unchanged from the value it contained after the previous call.

You can specify the RETURN-CODE special register in a function wherever an integer argument is allowed.

For more information on passing return code information, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

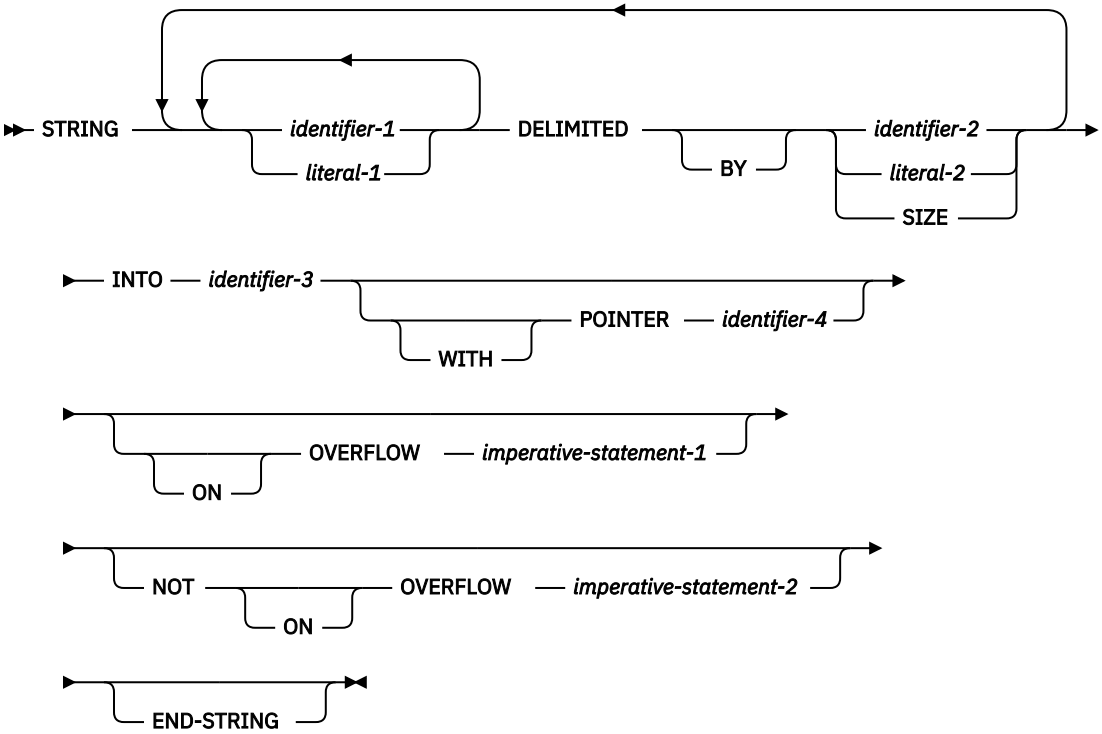
[End of IBM Extension]

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.

STRING Statement - 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 without the ALL literal. When a figurative constant is specified, it is considered a 1-character nonnumeric literal.

[IBM Extension]

identifier-1 through identifier-3
Can not be external floating-point items.

[End of IBM Extension]

[IBM Extension]

If one of **identifier-1**, **identifier-2**, or **identifier-3** is a DBCS data item, then all of them must be DBCS data items and all literals must be DBCS literals.

If one of **identifier-1**, **identifier-2**, or **identifier-3** is a national data item, then all of them must be national data items.

If one of **literal-1** or **literal-2** is a DBCS literal, then they must both be DBCS literals and identifier-1 through identifier-3 must be DBCS data items.

SPACE is the only figurative constant allowed for DBCS items.

[End of IBM Extension]

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.

[IBM Extension] It must not represent an external floating-point data item. [End of IBM Extension]

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.

[IBM Extension] When identifier-3 is a DBCS data item, identifier-4 indicates the relative DBCS character position in the receiving field. [End of IBM Extension]

ON OVERFLOW Phrases

Control is transferred to imperative-statement-1 when the pointer value (explicit or implicit):

- Is zero or less than 1
- Exceeds a value equal to the length of the receiving field

When any of the above conditions occur, an overflow condition exists, and no more data is transferred. The STRING operation is terminated and, if the ON OVERFLOW phrase is specified, control is transferred to imperative-statement-1. Otherwise, control is transferred to the end of the STRING statement. The NOT ON OVERFLOW statement, if specified, is ignored.

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 an overflow condition does not occur during the execution of a STRING statement, then control is transferred to the end of the STRING statement. If an overflow condition does not occur and the NOT ON OVERFLOW phrase is specified, control is transferred to imperative-statement-2. The ON OVERFLOW phrase, if specified, is ignored.

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, or the initial value of POINTER is less than 1.

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 243.

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 338).
- 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.

Subscripting, reference modification, variable-length calculations, or function evaluations are 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, reference modifier, or function argument in the STRING statement, or affects the length or location of any identifiers of 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.

If identifier-1 or identifier-2 occupy the same storage area as identifier-3 or identifier-4, or if identifier-3 and identifier-4 occupy the same storage area, the result of the execution of the STRING statement is undefined.

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 24 on page 419.

```
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
```

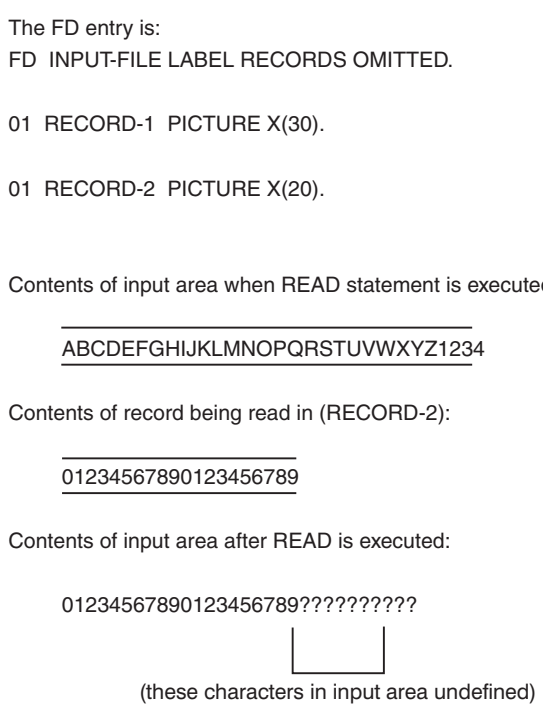


Figure 24. Results of STRING Statement Execution

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.␣SMITH␣␣␣␣␣
444␣SPRING␣ST.,␣CHICAGO,␣ILL.␣␣␣␣␣
A14275
$4,736.85
$2,400.00
09/22/76
$2,336.85
09/09/94

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 25 on page 421.
- LINE-POS contains the value 82.

Note: One STRING statement can be written instead of a series of MOVE statements.

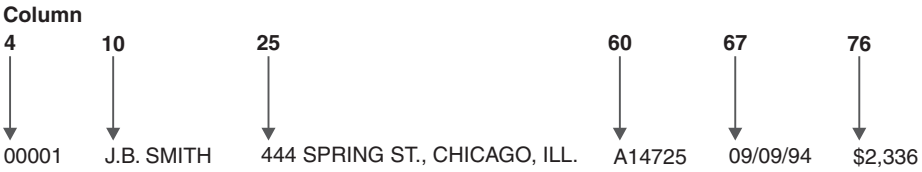
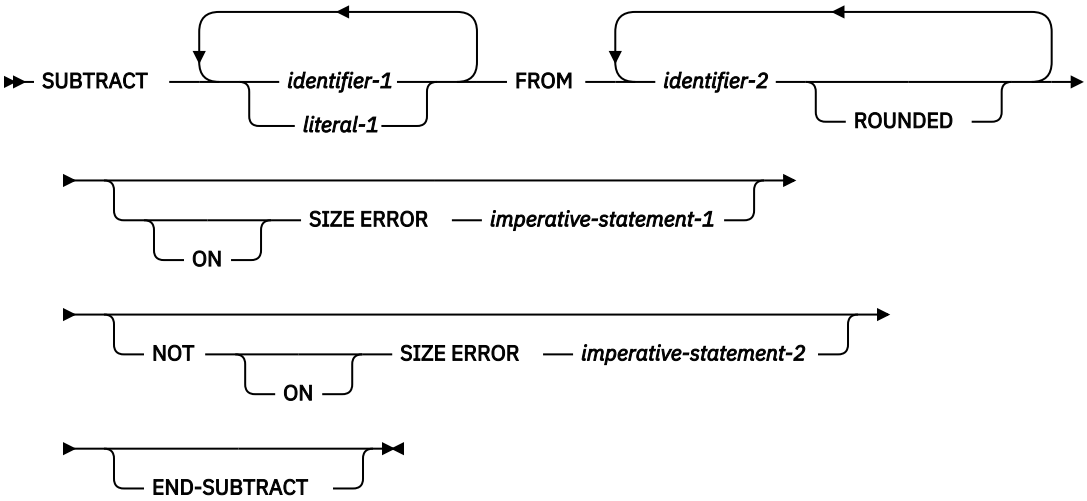


Figure 25. STRING Statement Example Output Data

SUBTRACT Statement

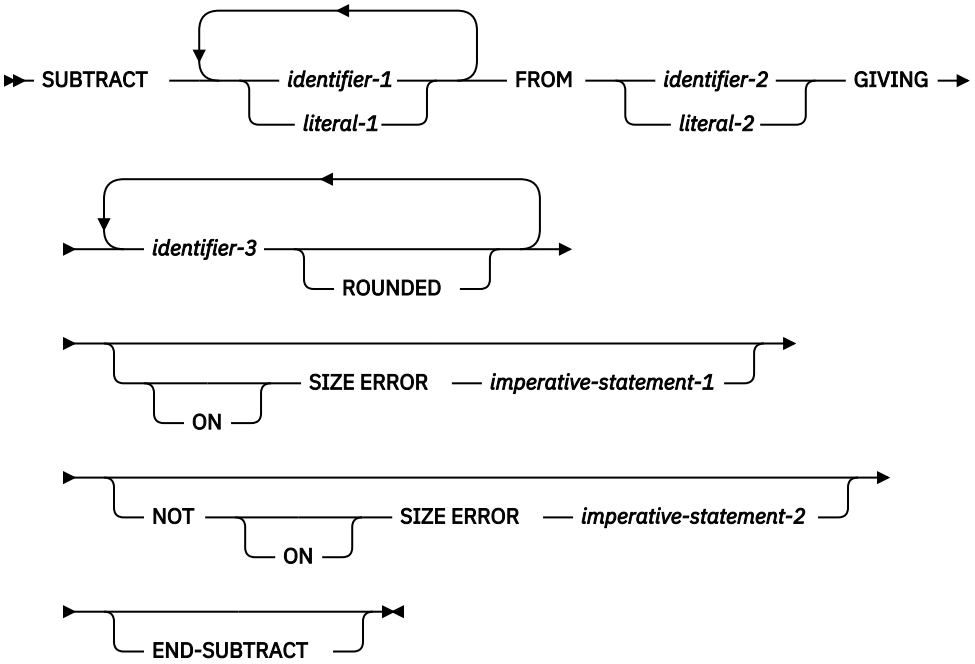
The SUBTRACT statement subtracts one numeric item, or the sum of two or more numeric items, from one or more numeric items, and stores the results.

SUBTRACT Statement - Format 1



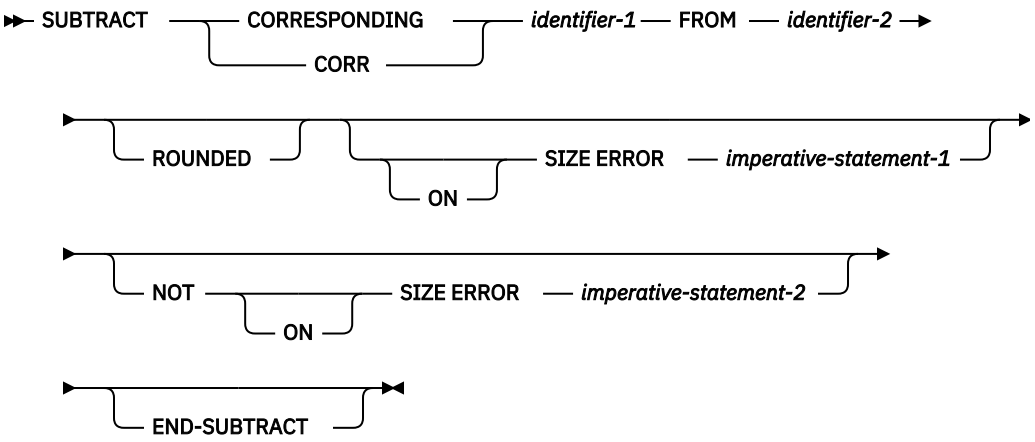
In Format 1, identifiers or literals preceding the keyword FROM are added together, and this initial sum is subtracted from and stored in identifier-2. The initial sum is then subtracted from and stored in each successive occurrence of identifier-2, in the left-to-right order in which identifier-2 is specified.

SUBTRACT Statement - Format 2 - GIVING



In Format 2, all identifiers or literals preceding the keyword FROM are added together and this sum is subtracted from identifier-2 or literal-2. The result of the subtraction is stored in the data item referenced by identifier-3. Identifier-2 or literal-2 remains unchanged.

SUBTRACT Statement - Format 3 - CORRESPONDING



In 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.

For all Formats:

identifier-1, identifier-2, identifier-3
In Formats 1 and 2, identifier-1 and identifier-2 must be elementary numeric items.

In Format 2, each identifier-3 following the word GIVING must be a numeric or numeric-edited elementary item.

In Format 3, identifier-1 must be a group item.

literal-1, literal-2
Must be a numeric literal.

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. For more information on the composite of operands, see the “Size of Operands” on page 247.

[IBM Extension] Floating-point data items and literals can be used anywhere numeric data items and literals can be specified. [End of IBM Extension]

ROUNDED Phrase

For information on the ROUNDED phrase, and for operand considerations, see “ROUNDED Phrase” on page 246.

SIZE ERROR Phrases

For information on the SIZE ERROR phrases, and for operand considerations, see “SIZE ERROR Phrases” on page 246.

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.

END-SUBTRACT Phrase

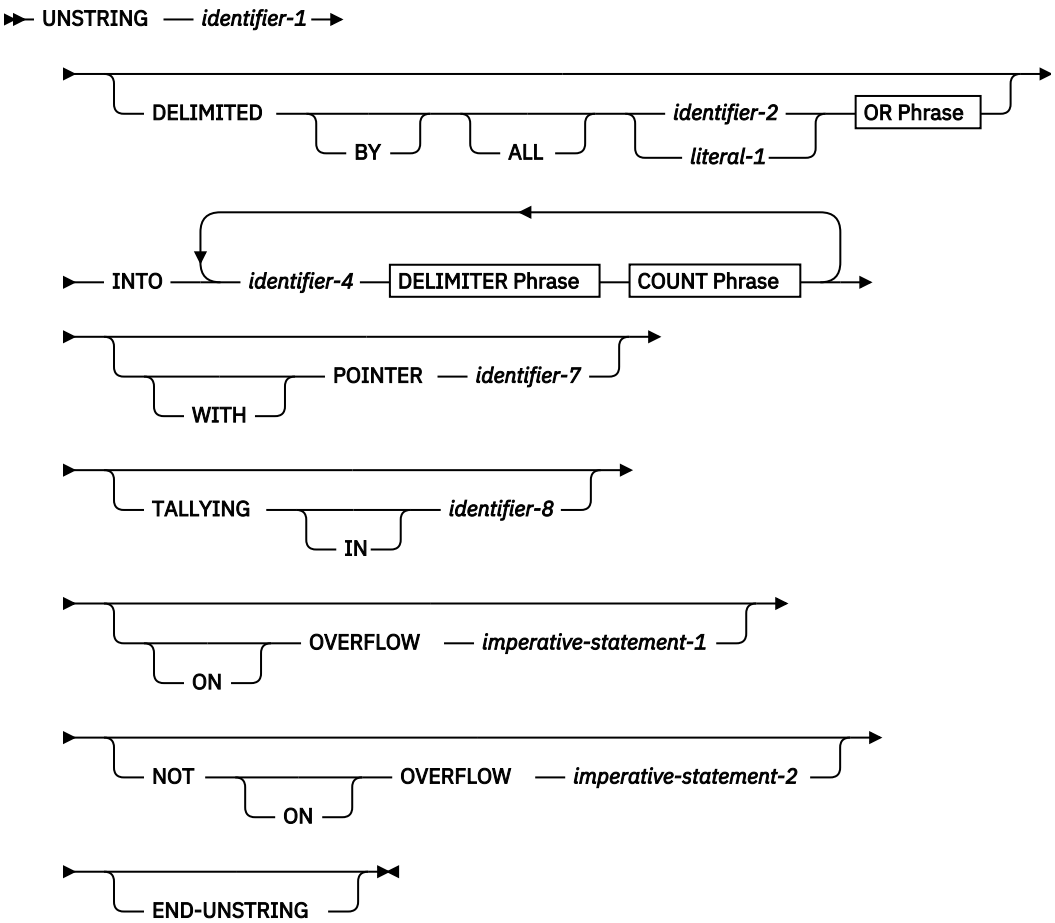
This explicit scope terminator delimits the scope of the SUBTRACT statement. END-SUBTRACT converts a conditional SUBTRACT statement into an imperative statement so that it can be nested in another conditional statement.

For more information, see “Delimited Scope Statements” on page 243.

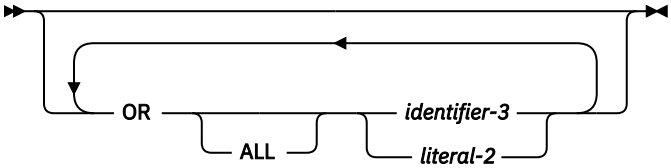
UNSTRING Statement

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

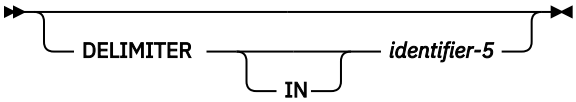
One UNSTRING statement can be written instead of a series of MOVE statements.



OR Phrase

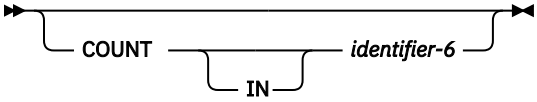


DELIMITER Phrase



UNSTRING Statement - Format

COUNT Phrase



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.

[IBM Extension] Identifier-1 can be a DBCS or national data-item. [End of IBM Extension]

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.

[IBM Extension] If either one is a DBCS item, then both must be DBCS items. If either one is a national item, then both must be national items. [End of IBM Extension]

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.

[IBM Extension] If either literal is a DBCS literal, then both must be DBCS literals. If either literal is a national literal, then both must be national literals. The figurative constant SPACE can be used as a DBCS or national literal. [End of IBM Extension]

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.

[IBM Extension] If ALL is used with a DBCS or national identifier or literal, the delimiting characters in the sending field are treated as the same type (DBCS or national). They are moved according to the rules of the MOVE statement. [End of IBM Extension]

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.

[IBM Extension] If the current receiving field is a national item, it is padded or filled as needed with the 'national to national' padding character specified in the NTLPADCHAR compiler or PROCESS statement option. [End of IBM Extension]

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).

[IBM Extension] Identifier-4 cannot be defined as a floating-point item. [End of IBM Extension]

[IBM Extension] Identifier-4 can be a DBCS or national data-item. [End of IBM Extension]

DELIMITER IN

Identifier-5 represents the **delimiter receiving fields**. Identifier-5 must be alphanumeric.

[IBM Extension] Identifier-5 can be a DBCS or national data-item. [End of IBM Extension]

The DELIMITER IN phrase can be specified only if the DELIMITED BY phrase is specified. The identifiers must not be defined as alphanumeric edited or numeric edited items. If no delimiter is found in identifier-1, then identifier-5 is space filled.

[IBM Extension] If identifier-5 is a national item, it is padded or filled as needed with the 'national to national' padding character specified in the NTLPADCHAR compiler or PROCESS statement option. [End of IBM Extension]

COUNT IN

Identifier-6, an integer data-item defined without the symbol P in the PICTURE string, is the **data-count field** for each data transfer. Each field holds the count of examined characters in the sending field, terminated by the delimiters or the end of the sending field, for the move to this receiving field; the delimiters are not included in this count.

[IBM Extension] When identifier-1 (the sending field) is a DBCS or national data item, identifier-6 indicates the number of characters (not the number of bytes) examined in the sending field. [End of IBM Extension]

The COUNT IN phrase must not be specified unless the DELIMITED BY phrase is specified.

POINTER Phrase

identifier-7

Identifier-7, an integer data-item defined without the symbol P in the PICTURE string, 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

Identifier-8 is the **field-count field**, initialized by the user through an integer data-item defined without the symbol P in the PICTURE string, and increased by the number of data receiving fields acted upon in this execution of the UNSTRING statement.

ON OVERFLOW Phrases

Imperative-statement-1 is 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:

1. An overflow condition exists, and no more data is transferred
2. The UNSTRING operation is terminated
3. The NOT ON OVERFLOW phrase, if specified, is ignored
4. 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 conditions that would cause an overflow condition are not encountered, the ON OVERFLOW phrase, if specified, is ignored. If the NOT ON OVERFLOW phrase is specified, control is transferred to imperative-statement-2; otherwise, control is transferred to the end of the UNSTRING statement.

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 243.

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 338).
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, reference modification, variable-length calculations, or function evaluations are 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 26 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
          ID-R2  DELIMITER  IN ID-D2
          ID-R3  DELIMITER  IN ID-D3    COUNT  IN ID-C3
          ID-R4  COUNT  IN ID-C4
     WITH POINTER ID-P
     TALLYING  IN  ID-T
     ON OVERFLOW  GO TO  OFLOW-EXIT.
```

(All the data receiving fields are defined as alphanumeric)

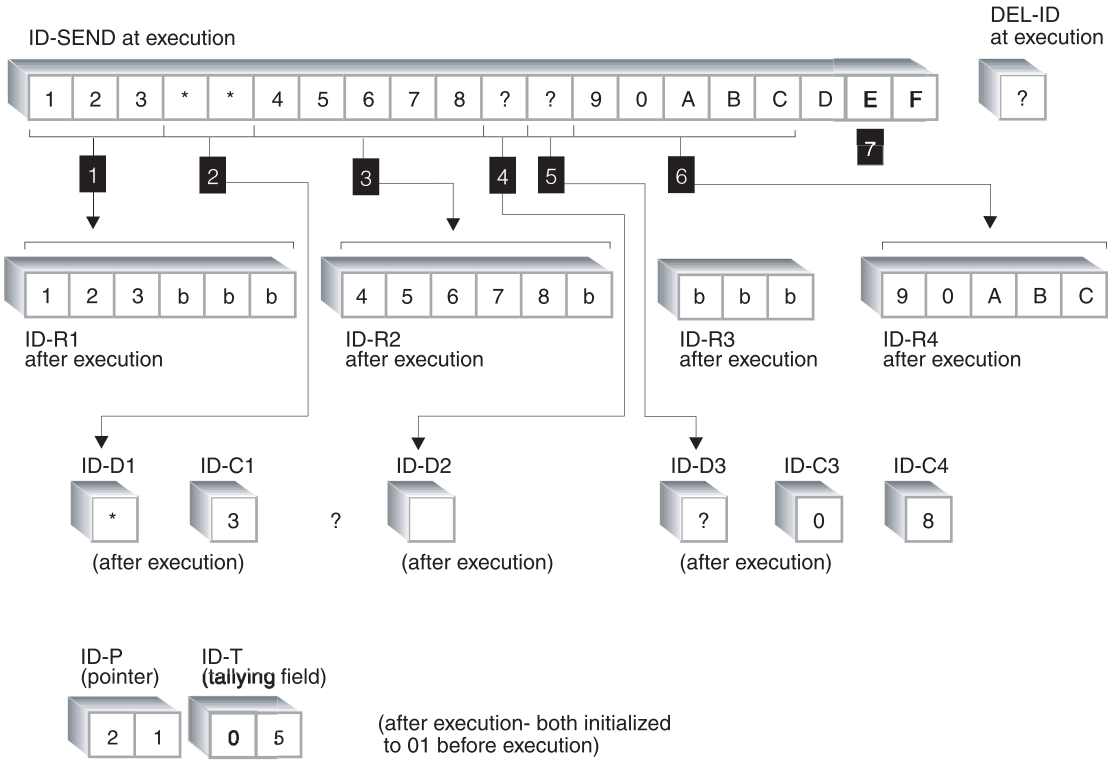


Figure 26. Results of UNSTRING Statement Execution

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.

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,
 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 27 on page 430.

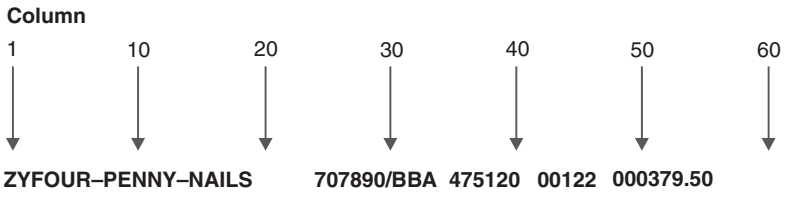


Figure 27. UNSTRING Statement Example–Input Data

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.

Note: One UNSTRING statement can be written instead of a series of MOVE statements.

WRITE Statement

The WRITE statement releases a 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 (device types TAPEFILE, DISK, or DATABASE) mode.

[IBM Extension]

- Format 3 - FORMATFILE
- Format 4 - TRANSACTION (Nonsubfile)
- Format 5 - TRANSACTION (Subfile)

[End of IBM Extension]

[IBM Extension]

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.

For more information on this command, see the *CL and APIs* section of the *Programming* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

[End of IBM Extension]

Sequential Files

The ADVANCING and END-OF-PAGE phrases control the vertical positioning of each line on a printed page. 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.

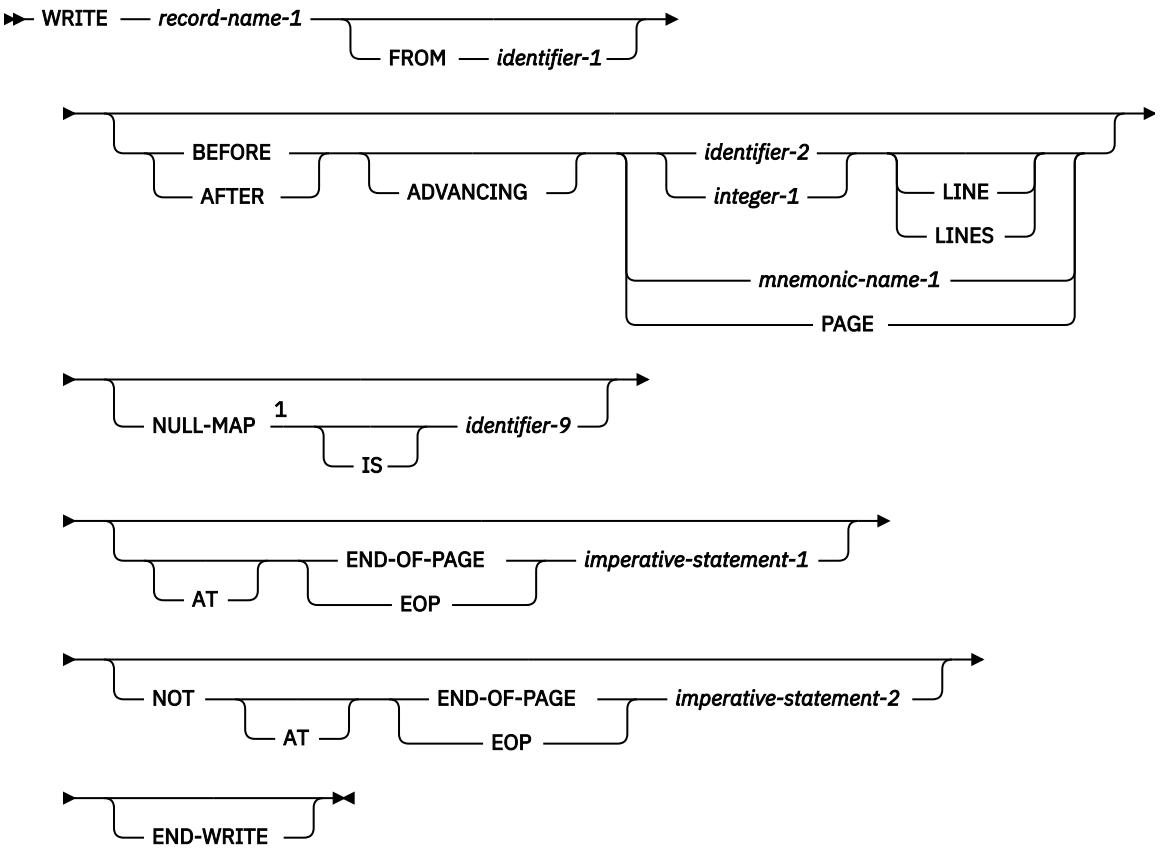
Note: The ADVANCING PAGE and END-OF-PAGE phrases must not both be specified in a single WRITE statement.

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 534.

For sequential files on device type TAPEFILE or DISKETTE, when end-of-volume is recognized for a multivolume OUTPUT file, the WRITE statement processes the following operations in 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.



WRITE Statement - Format 1 - Sequential Files

Notes:

[1] IBM Extension.

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.

You must ensure that the size of an OCCURS DEPENDING ON (ODO) array has been set to a valid numeric value before doing a WRITE of a record containing the ODO array.

[IBM Extension] Record-name-1 can be a floating-point data item. [End of IBM Extension]

[IBM Extension] Record-name-1 can define a DBCS data item. Identifier-1 must be a DBCS data-item if record-name-1 is a DBCS data item. [End of IBM Extension]

FROM

When FROM is specified, the result is the same as:

```
MOVE identifier-1 TO record-name-1
WRITE record-name-1
```

The move is performed according to the rules of the MOVE statement, without the CORRESPONDING phrase.

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 250.)

identifier-1

Must be an alphanumeric or numeric edited data item. Data is transferred from this field to the receiving fields.

Identifier-1 can be the name of an alphanumeric or DBCS function identifier.

[IBM Extension] Identifier-1 can be a floating-point or date-time data item. [End of IBM Extension]

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 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 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 185 and “USAGE Clause” on page 201.)

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 cannot 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. It only applies to device type PRINTER. 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 be an integer data item.
4. When integer-1 is specified, the page is advanced the number of lines equal to the value of integer-1.
5. Integer-1 or the value in identifier-2 may be zero.
6. When mnemonic-name is specified, a system-specific action takes place. Mnemonic-name must be equated with environment-name-1 in the SPECIAL-NAMES paragraph (valid environment-names are listed in Table 2 on page 80). For more information on acceptable values for mnemonic-name, see “SPECIAL-NAMES Paragraph” on page 78.
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:

   a. If ADVANCING PAGE is specified, LINAGE-COUNTER is reset to 1.
   b. If ADVANCING identifier-2 or integer-1 is specified, LINAGE-COUNTER is increased by the value in identifier-2 or integer-1.
   c. If the ADVANCING phrase is omitted, LINAGE-COUNTER is increased by 1.
   d. 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.

[IBM Extension] NULL-MAP IS Phrase

Refer to the description supplied for this phrase on page NULL-MAP IS Phrase.

[End of IBM Extension]

END-OF-PAGE Phrase

When this phrase is specified (and the FD entry for this file contains a LINAGE clause), and the logical end of the printed page is reached during execution of the WRITE statement, the imperative-statement is executed.

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.

Special Considerations for Printer Files

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.

Special Considerations for FORMATFILEs

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.

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.

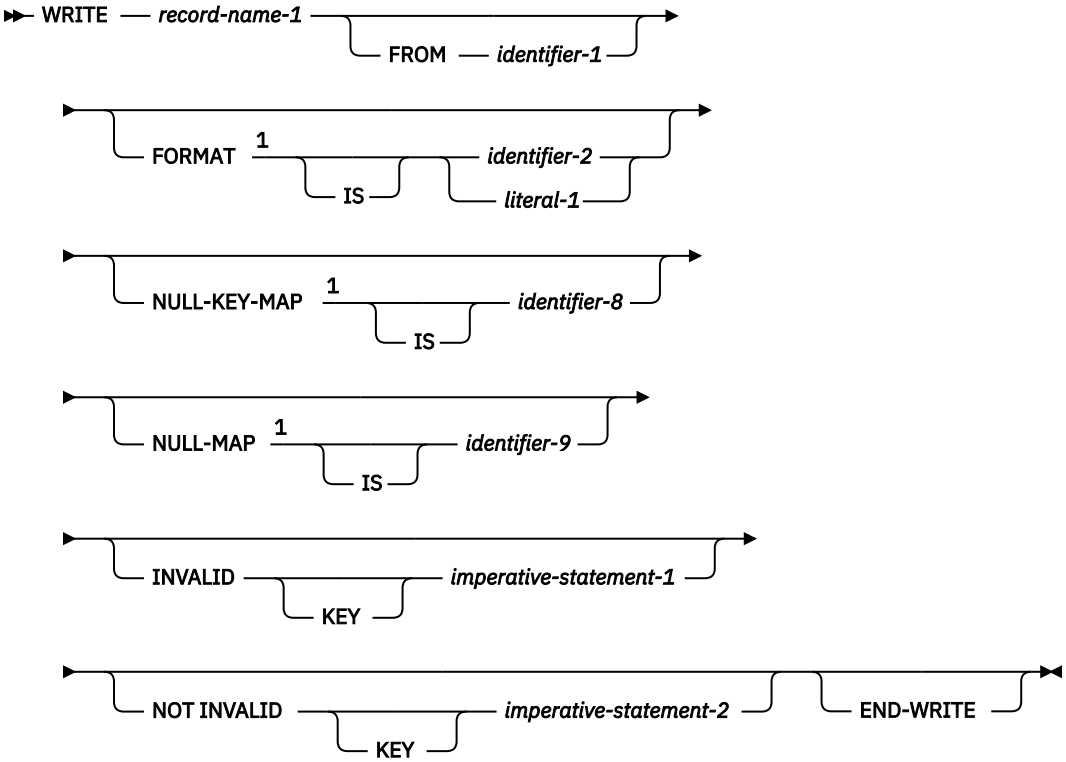
For more information, see "Delimited Scope Statements" on page 243.

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



WRITE - Format 2 - Indexed and Relative Files

Notes:

[1] IBM Extension

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.

You must ensure that the size of an OCCURS DEPENDING ON (ODO) array has been set to a valid numeric value before doing a WRITE of a record containing the ODO array.

In the case of relative files, only, 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 250.)

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.

Identifier-1 can be the name of an alphanumeric or DBCS function identifier.

Considerations When Writing Indexed Files

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. When the WRITE statement is processed, the system releases the record.

[IBM Extension] If the DUPLICATES phrase is specified, record key values for a format need not be unique (see "RECORD KEY Clause" on page 110). 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]

If records are written to an indexed file of fixed size when it has SEQUENTIAL access, is open for OUTPUT, and blocking is in effect (BLOCK CONTAINS clause is specified), the blocking factor will change to 1 at the point at which a block of records would cause the end-of-file to be reached.

If the ALTERNATE RECORD KEY clause is also specified in the File-Control entry, each alternate record key must be unique, unless the DUPLICATES phrase is specified. If the DUPLICATES phrase is specified, alternate record key values need not be unique.

The number of remaining records in the file at this moment is less than the number of records in a block.

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. If the FORMAT phrase is not specified on the I-O statement when indexed files are accessed in random access mode, the first format defined is used. When writing to a multiformat logical file, the format must be specified on the WRITE statement.

Considerations When Writing Relative Files

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 files opened in I-O mode, 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 (for example, using the CRTPF with the ALWDLT(*NO) parameter), the update operation on records must be allowed (that is, CRTPF with the ALWUPD(*YES) parameter).

[IBM Extension] FORMAT Phrase

Required if there is more than one record format for the file.

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.

Identifier-2, if specified, must be a an alphanumeric data item of 10 characters or less.

Literal-1, if specified, must be an uppercase character-string of 10 characters or less.

If the FORMAT phrase is not specified on the I-O statement when indexed files are accessed in random access mode, the first format defined is used.

[End of IBM Extension]

[IBM Extension] NULL-KEY-MAP IS Phrase

Refer to the description supplied for this phrase on page NULL-KEY-MAP IS Phrase.

[End of IBM Extension]

[IBM Extension] NULL-MAP IS Phrase

Refer to the description supplied for this phrase on page NULL-MAP IS Phrase.

[End of IBM Extension]

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.

For Relative files in Random or Dynamic access mode, an INVALID KEY condition exists when RELATIVE KEY specifies a record that already contains data.

For Indexed files in Random or Dynamic access mode, 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.

For Indexed files in Sequential access mode, an INVALID KEY condition exists when the values of the primary record keys of successive records are not in ascending order.

[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]

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 250.

NOT INVALID KEY Phrase

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.

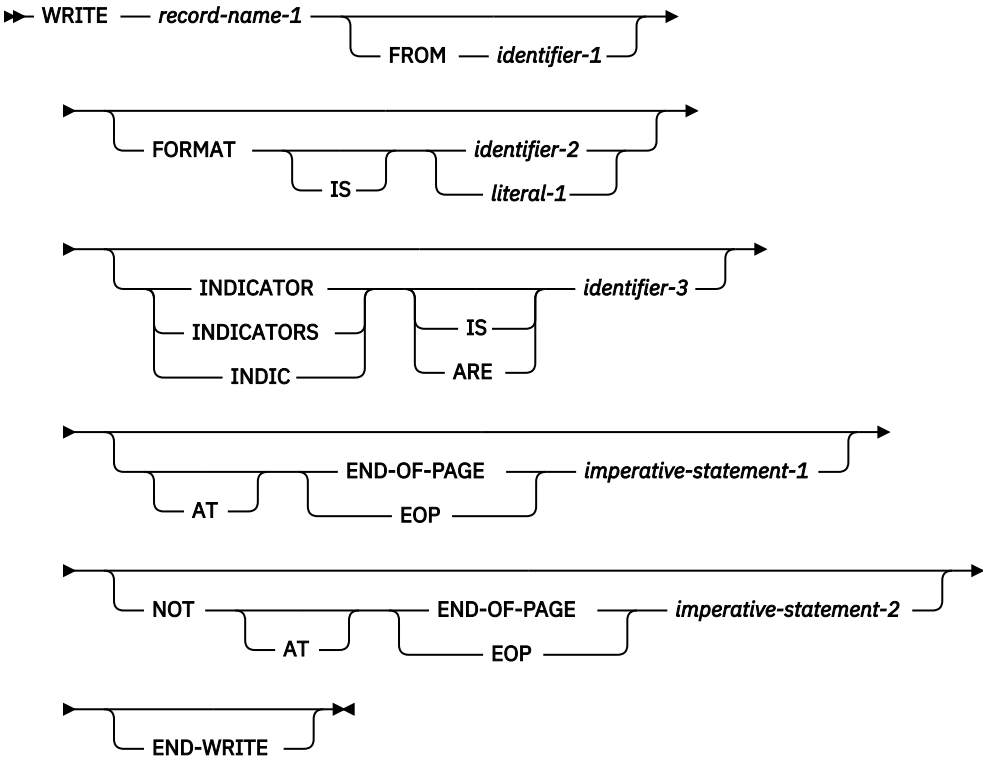
END-WRITE Phrase

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 243.

[IBM Extension] FORMATFILE

WRITE Statement - Format 3 - FORMATFILE



[End of IBM Extension]

FORMAT Phrase

Required if there is more than one record format for the file.

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.

Identifier-2, if specified, must be an alphanumeric data item of 10 characters or less.

Literal-1, if specified, must be an uppercase character-string of 10 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.

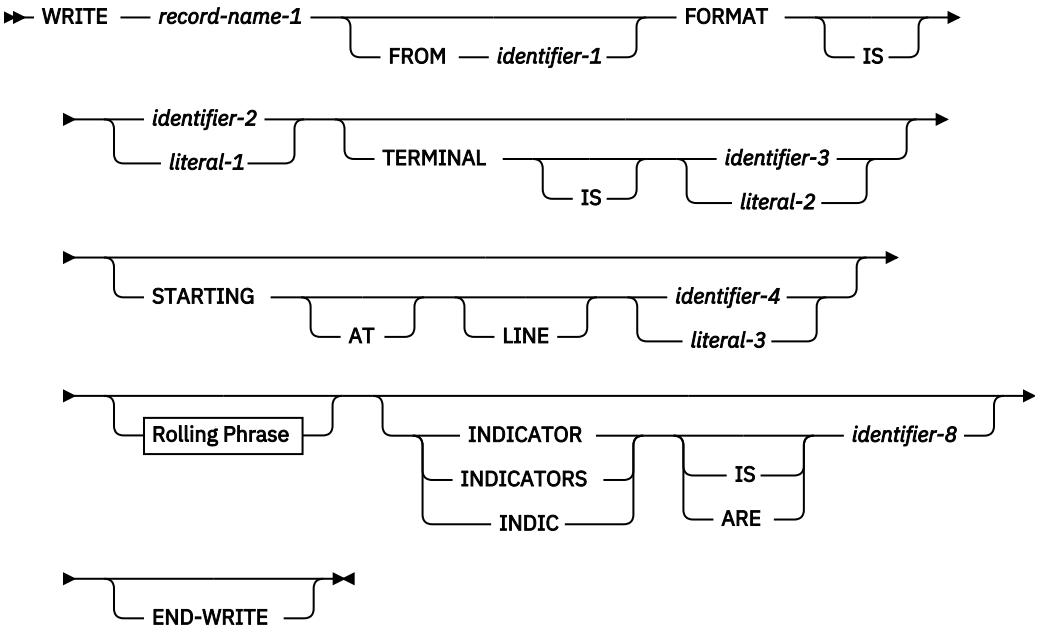
INDICATORS Phrase

Specifies which indicators are to be written when a data record is read. Indicators can be used to pass information about the data record and how it was entered into the program.

For detailed information on the INDICATORS phrase, refer to the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

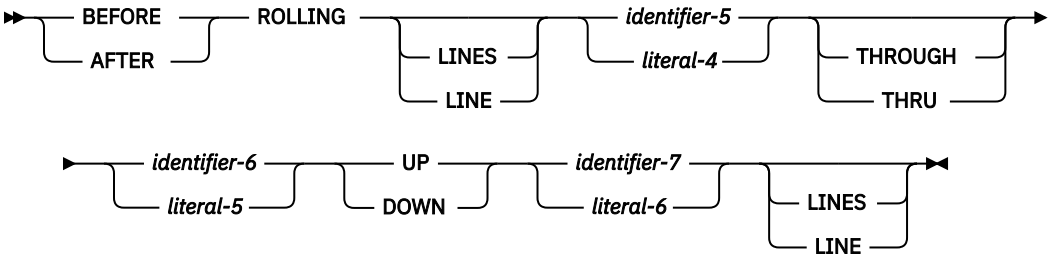
Identifier-3 must be either an elementary Boolean data item specified without the OCCURS clause or a group item that has elementary Boolean data items subordinate to it.

[IBM Extension] TRANSACTION (Nonsubfile)



WRITE - Format 4 - TRANSACTION (Nonsubfile)

Rolling Phrase



[End of IBM Extension]

FORMAT Phrase

Literal-1 or identifier-2 specifies the name of the record format to be written. 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 WRITE statement is executed as if the FORMAT phrase were omitted.

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

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 workstation 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 workstation 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.

For further information on SLNO(*VAR) and CLRL, see the *Db2 for i* section of the *Database and File Systems* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

ROLLING Phrase

The ROLLING phrase allows you to move lines displayed on the workstation 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 workstation 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.
- 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.

- 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.

For more information, see the *Db2 for i* section of the *Database and File Systems* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

INDICATORS Phrase

Specifies which indicators are to be used when a data record is written. Indicators can be used to pass information about the data record and how it was entered into the program.

For detailed information on the INDICATORS phrase, refer to Using Indicators with Transaction Files in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

Identifier-8 must be either an elementary Boolean data item specified without the OCCURS clause or a group item that has elementary Boolean data items subordinate to it.

Figure 28 on page 442 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 workstation screen. Part of FMT1 is rolled down 2 lines before FMT2 is written to the workstation screen.

Execution of the following WRITE statement causes part of FMT1 to be rolled down 2 lines, and FMT2 to be written to the workstation 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 workstation 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 workstation screen are returned to the program by a READ statement, only the input capable fields of FMT2 are returned.

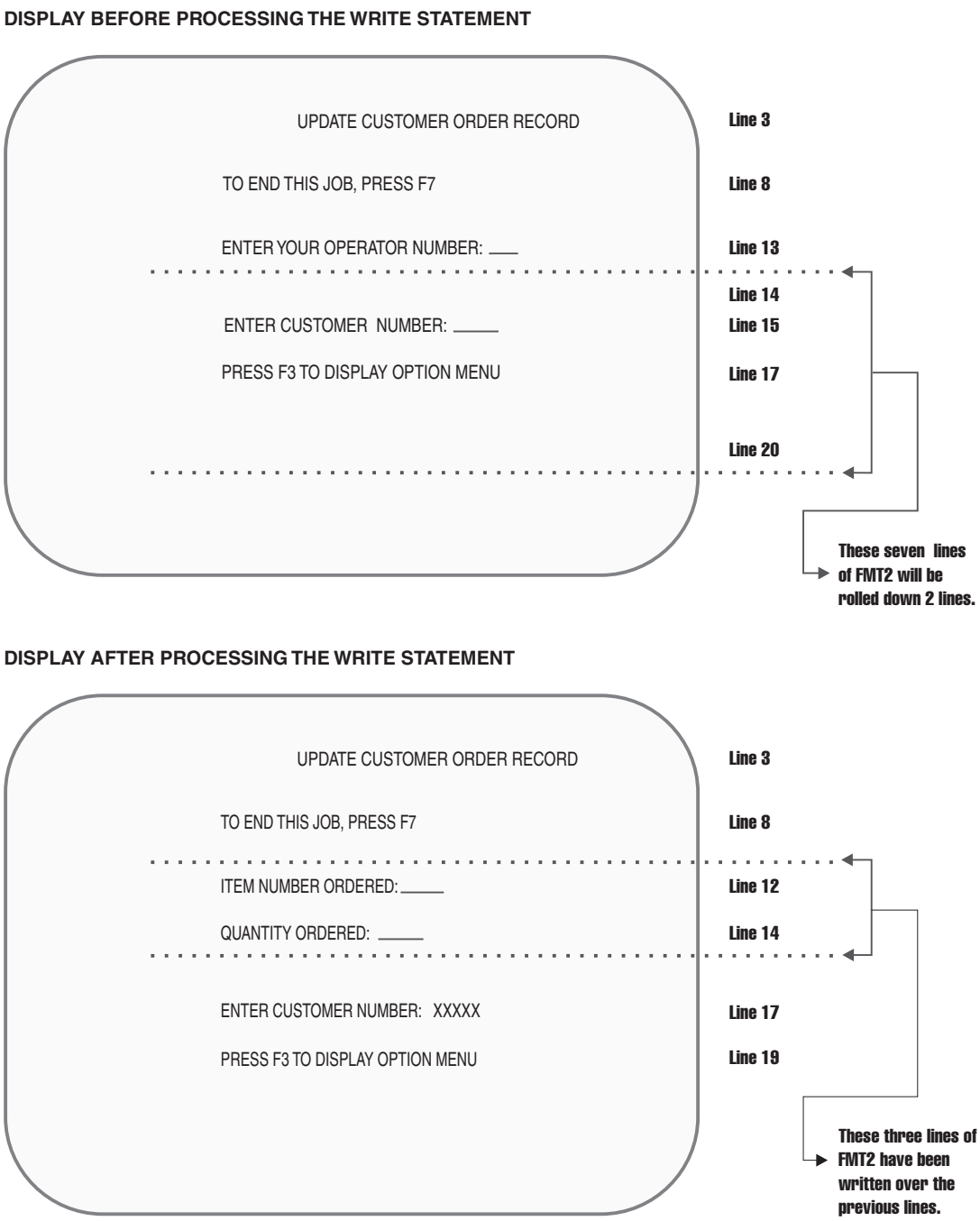
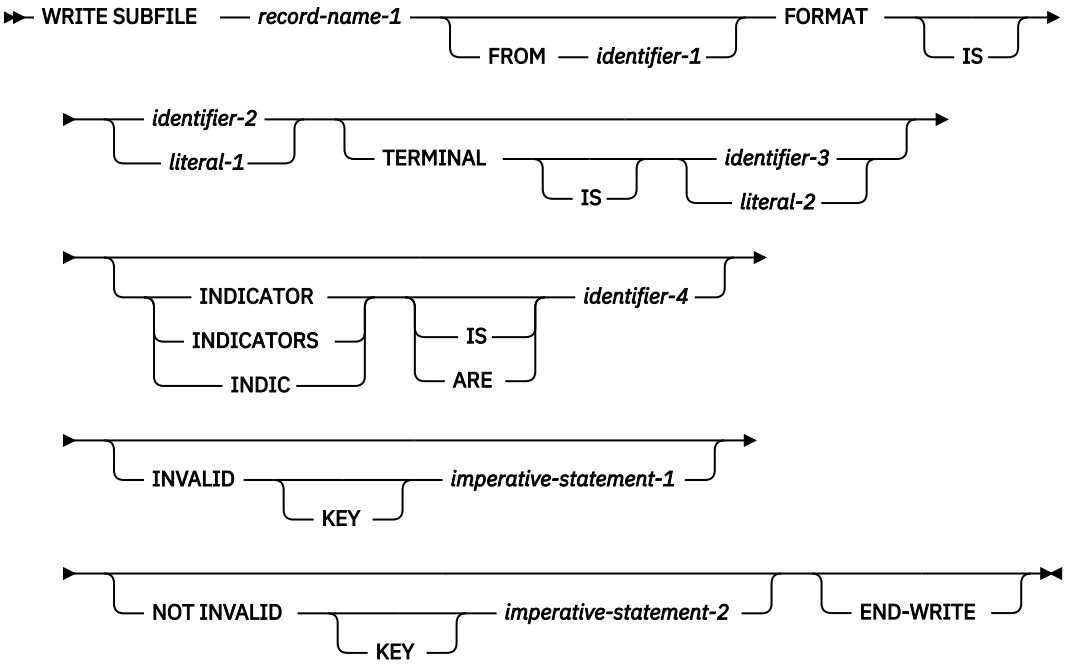


Figure 28. Example of ROLLING Operation

[IBM Extension] TRANSACTION (Subfile)

WRITE Statement - 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 be 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.

[End of IBM Extension]

INDICATORS Phrase

Specifies which indicators are to be used when a data record is written. Indicators can be used to pass information about the data record and how it was entered into the program.

For detailed information on the INDICATORS phrase, refer to Using Indicators with Transaction Files in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

Identifier-4 must be either an elementary Boolean data item specified without the OCCURS clause or a group item that has elementary Boolean data items subordinate to it.

TERMINAL Phrase

See Format 4 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 workstation 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 workstation 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

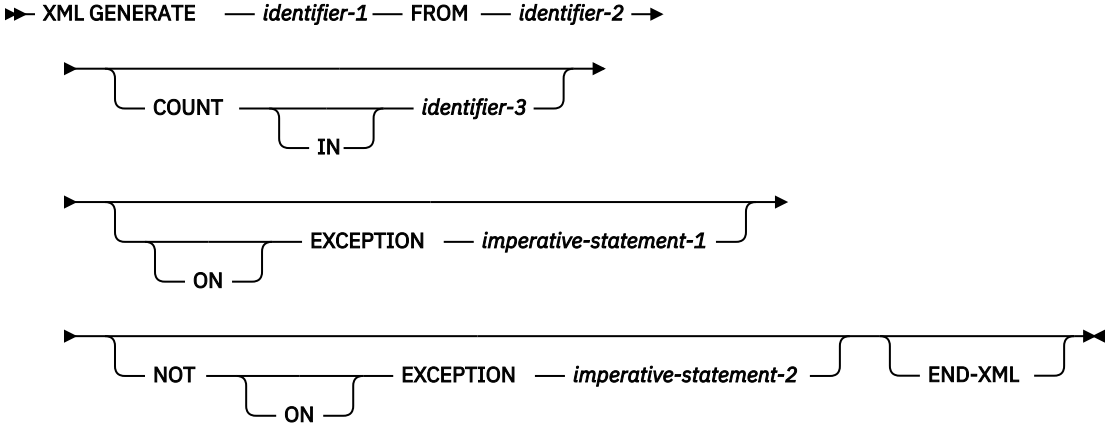
This explicit scope terminator serves to delimit the scope of the WRITE statement. END-WRITE permits a conditional WRITE statement to be nested in another conditional statement. END-WRITE may also be used with an imperative WRITE statement.

For more information, see "Delimited Scope Statements" on page 243.

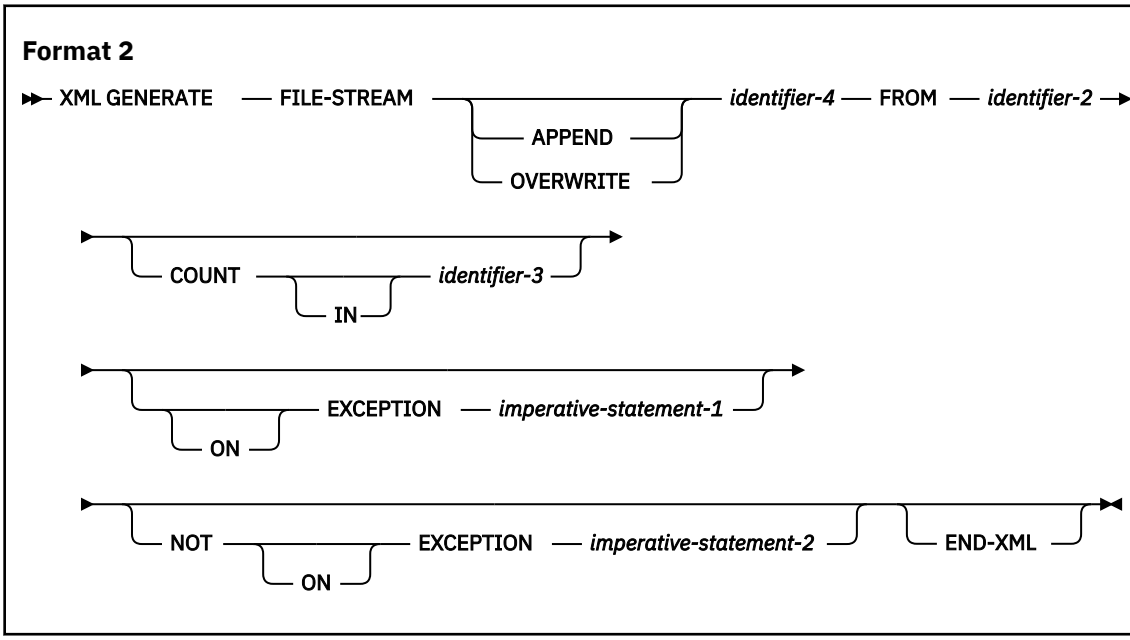
[IBM Extension] XML GENERATE Statement

The XML GENERATE statement converts data to XML format.

Format 1



Format 2



identifier-1

The receiving area for a generated XML document. *identifier-1* must reference one of the following:

- An elementary data item of category alphanumeric
- An alphanumeric group item
- An elementary data item of category national

When *identifier-1* references an alphanumeric group item, *identifier-1* is treated as though it were an elementary data item of category alphanumeric.

identifier-1 must not be described with the JUSTIFIED clause, and cannot be a function identifier. *identifier-1* can be subscripted or reference modified.

identifier-1 must not overlap *identifier-2* or *identifier-3*.

If *identifier-1* references a data item of category alphanumeric, the generated XML document is encoded with the CCSID specified by the PROCESS statement CCSID option d - XML GENERATE single-byte data CCSID in effect when the source code was compiled. If the CCSID in effect is 65535, the job default CCSID at run time will be used.

If *identifier-1* references a data item of category national, the generated XML document is encoded in UCS-2. If PROCESS statement CCSID option d specifies a National CCSID, that CCSID is used. Otherwise, the CCSID specified by the NTLCCSID PROCESS option is used. A byte order mark is not generated.

identifier-1 must reference a data item of category national if the generated XML includes data from *identifier-2* for:

- Any data item of class national or class DBCS
- Any data item with a DBCS name (that is, a data item whose name contains DBCS characters)

identifier-1 must be large enough to contain the generated XML document. Typically, it should be from five to eight times the size of *identifier-2*, depending on the length of the data-name or data-names within *identifier-2*. If *identifier-1* is not large enough, an error condition exists at the end of the XML GENERATE statement.

identifier-2

The group or elementary data item to be converted to XML format.

identifier-2 cannot be a function identifier or be reference modified, but it can be subscripted.

identifier-2 must not overlap with *identifier-1* or *identifier-3*.

identifier-2 must not specify the RENAMES clause.

The following data items specified by *identifier-2* are ignored by the XML GENERATE statement:

- Any unnamed elementary data items or elementary FILLER data items
- Any slack bytes inserted for SYNCHRONIZED items
- Any data item subordinate to *identifier-2* that is described with the REDEFINES clause or that is subordinate to such a redefining item
- Any data item subordinate to *identifier-2* that is described with the RENAMES clause
- Any group data item all of whose subordinate data items are ignored

All data items specified by *identifier-2* that are not ignored according to the rules above must satisfy the following conditions:

- Each elementary data item must either have class alphabetic, alphanumeric, numeric, or national, or be an index data item. (That is, no elementary data item can be described with the USAGE POINTER or USAGE PROCEDURE-POINTER phrase.)
- There must be at least one such elementary data item.

- Each non-FILLER data-name must be unique within any immediately superordinate group data item.
- Any DBCS data-names, when converted to Unicode, must be legal as names in the XML specification, version 1.0.

For example, given the following data declaration:

```
01 STRUCT.
  02 STAT PIC X(4).
  02 IN-AREA PIC X(100).
  02 OK-AREA REDEFINES IN-AREA.
    03 FLAGS PIC X.
    03 PIC X(3).
    03 COUNTER USAGE COMP PIC S9(9).
    03 ASFNPTR REDEFINES COUNTER USAGE PROCEDURE-POINTER.
    03 UNREFERENCED PIC X(92).
  02 NG-AREA1 REDEFINES IN-AREA.
    03 FLAGS PIC X.
    03 PIC X(3).
    03 PTR USAGE POINTER.
    03 ASNUM REDEFINES PTR USAGE COMP PIC S9(9).
    03 PIC X(92).
  02 NG-AREA2 REDEFINES IN-AREA.
    03 FN-CODE PIC X.
    03 UNREFERENCED PIC X(3).
    03 QTYONHAND USAGE BINARY PIC 9(5).
    03 DESC USAGE NATIONAL PIC N(40).
    03 UNREFERENCED PIC X(12).
```

The following data items can be specified as *identifier-2*:

- STRUCT, of which subordinate data items STAT and IN-AREA would be converted to XML format. (OK-AREA, NG-AREA1, and NG-AREA2 are ignored because they specify the REDEFINES clause.)
- OK-AREA, of which subordinate data items FLAGS, COUNTER, and UNREFERENCED would be converted. (The item whose data description entry specifies `03 PIC X(3)` is ignored because it is an elementary FILLER data item. ASFNPTR is ignored because it specifies the REDEFINES clause.)
- Any of the elementary data items that are subordinate to STRUCT except:
  - ASFNPTR or PTR (disallowed usage)
  - `UNREFERENCED OF NG-AREA2` (nonunique names for data items that are otherwise eligible)
  - Any FILLER data items

The following data items cannot be specified as *identifier-2*:

- NG-AREA1, because subordinate data item PTR specifies USAGE POINTER but does not specify the REDEFINES clause. (PTR would be ignored if it specified the REDEFINES clause.)
- NG-AREA2, because subordinate elementary data items have the nonunique name UNREFERENCED.

COUNT IN

If the COUNT IN phrase is specified, *identifier-3* contains (after execution of the XML GENERATE statement) the count of generated XML character positions. If *identifier-1* (the receiver) has category national, the count is in national character positions (UCS-2 character encoding units). Otherwise, the count is in bytes.

identifier-3

The data count field. Must be an integer data item defined without the symbol P in its picture string.

identifier-3 must not overlap *identifier-1* or *identifier-2*.

FILE-STREAM phrase

When the FILE-STREAM phrase is specified, the converted XML data will be saved to an IFS file that is specified by identifier-4. The XML file is encoded using the CCSID:

- Unicode UCS-2, if the generated XML includes (as described under "identifier-1") data from identifier-2 for:
  - Any national data item or DBCS data item
  - Any data item with an DBCS name
- Otherwise, the CCSID specified in the PROCESS statement CCSID option d (XML GENERATE single-byte data output CCSID, default is JOBRUN). The CCSID used must be one of the single-byte character set CCSIDs listed in "Coded character sets for XML documents" on page 453.

When no APPEND or OVERWRITE phrase is used, a new file will be created with the XML file encoding CCSID and the converted XML data will be saved into it. If a file with the same name exists when running a program, XML generation stops and the special register XML-CODE contains an exception code representing this error.

If APPEND phrase is used, the converted XML data will be appended to the existing file when the file has the XML file encoding CCSID; otherwise XML generation stops and the special register XML-CODE contains an exception code representing this error.

However, if PROCESS option XMLGEN(KEEPFILEOPEN) has been specified and the IFS file is currently open, then specifying XML GENERATE without the APPEND or OVERWRITE phrase can be used to close the IFS file, and no error or exception code will be issued

If OVERWRITE phrase is used, the existing file will be replaced by a new file with the XML file encoding CCSID; the converted XML data will be saved into the new file.

Any other file operation errors except those mentioned above will trigger a runtime inquiry message including file operation error message. If "G" is answered to continue the operation, an exception code will be set in special register XML-CODE.

identifier-4
Is the IFS file name field, and must be an alphabetic or alphanumeric data item. It contains the path name of the IFS file which will hold the converted XML contents.

ON EXCEPTION
An exception condition exists when an error occurs during generation of the XML document, for example if *identifier-1* is not large enough to contain the generated XML document. In this case, XML generation stops and the content of the receiver, *identifier-1*, is undefined. If the COUNT IN phrase is specified, *identifier-3* contains the number of character positions that were generated, which can range from 0 to the length of *identifier-1*.

If the ON EXCEPTION phrase is specified, control is transferred to *imperative-statement-1*. If the ON EXCEPTION phrase is not specified, the NOT ON EXCEPTION phrase, if any, is ignored, and control is transferred to the end of the XML GENERATE statement. Special register XML-CODE contains an exception code, as detailed in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

NOT ON EXCEPTION
If an exception condition does not occur during generation of the XML document, control is passed to *imperative-statement-2*, if specified, otherwise to the end of the XML GENERATE statement. The ON EXCEPTION phrase, if specified, is ignored. Special register XML-CODE contains zero after execution of the XML GENERATE statement.

END-XML phrase
This explicit scope terminator delimits the scope of XML GENERATE or XML PARSE statements. END-XML permits a conditional XML GENERATE or XML PARSE statement (that is, an XML GENERATE or XML PARSE statement that specifies the ON EXCEPTION or NOT ON EXCEPTION phrase) to be nested in another conditional statement.

The scope of a conditional XML GENERATE or XML PARSE statement can be terminated by:

- An END-XML phrase at the same level of nesting
- A separator period

END-XML can also be used with an XML GENERATE or XML PARSE statement that does not specify either the ON EXCEPTION or the NOT ON EXCEPTION phrase.

[End of IBM Extension]

Nested XML GENERATE or XML PARSE statements

When a given XML GENERATE or XML PARSE statement appears as *imperative-statement-1* or *imperative-statement-2*, or as part of *imperative-statement-1* or *imperative-statement-2* of another XML GENERATE or XML PARSE statement, that given XML GENERATE or XML PARSE statement is a *nested* XML GENERATE or XML PARSE statement.

Nested XML GENERATE or XML PARSE statements are considered to be matched XML GENERATE and END-XML, or XML PARSE and END-XML combinations proceeding from left to right. Thus, any END-XML phrase that is encountered is matched with the nearest preceding XML GENERATE or XML PARSE statement that has not been implicitly or explicitly terminated.

Operation of XML GENERATE

The content of each eligible elementary data item within *identifier-2* is converted to character format as described under "Format conversion of elementary data" on page 448 and "Trimming of generated XML data" on page 449. Only the first definition of each storage area is processed. Redefinitions of data items are not included. Data items that are effectively defined by the RENAMES clause are also not included.

The converted content is then inserted as element character content in XML markup. The XML element names are derived from the data-names within *identifier-2* as described under "XML element name formation" on page 450. The names of group items that contain the selected elementary items are retained as parent elements. No extra white space (new lines, indentation, and so forth) is inserted to make the generated XML more readable. An XML declaration is not generated.

If the receiving area specified by *identifier-1* is not large enough to contain the resulting XML document, an error condition exists. See the description of the ON EXCEPTION phrase above for details.

If *identifier-1* is longer than the generated XML document, only that part of *identifier-1* in which XML is generated is changed. The rest of *identifier-1* contains the data that was present before this execution of the XML GENERATE statement. To avoid referring to that data, either initialize *identifier-1* to spaces before the XML GENERATE statement or specify the COUNT IN phrase.

If the COUNT IN phrase is specified, *identifier-3* contains (after execution of the XML GENERATE statement) the total number of character positions (UCS-2 encoding units or bytes) that were generated. You can use *identifier-3* as a reference modification length field to refer to the part of *identifier-2* that contains the generated XML document.

After execution of the XML GENERATE statement, special register XML-CODE contains either zero, which indicates successful completion, or a nonzero exception code. (See also the *ILE COBOL Programming Guide* for details.)

The XML PARSE statement also uses special register XML-CODE. Therefore if you code an XML GENERATE statement in the processing procedure of an XML PARSE statement, save the value of XML-CODE before that XML GENERATE statement executes and restore the saved value after the XML GENERATE statement terminates.

Format conversion of elementary data

Elementary data items are converted to character format depending on the type of the data item:

- Data items of category alphabetic, alphanumeric, alphanumeric-edited, DBCS, external floating-point, national, and numeric-edited are not converted.
- Fixed-point numeric data items other than COMP-5 data items or binary data items compiled with the NOSTDTRUNC compiler option are converted as if they were moved to a numeric-edited item that has:

  - As many integer positions as the numeric item has, but with at least one integer position
  - An explicit decimal point, if the numeric item has at least one decimal position
  - The same number of decimal positions as the numeric item has
  - A leading '-' picture symbol if the data item is signed (has an S in its PICTURE clause)
- COMP-5 data items and binary data items compiled with the NOSTDTRUNC compiler option are converted in the same way as the other fixed-point numeric items, except for the number of integer positions. The number of integer positions is computed depending on the number of '9' symbols in the picture character string as follows:
  - 5 minus the number of decimal places, if the data item has 1 to 4 '9' picture symbols
  - 10 minus the number of decimal places, if the data item has 5 to 9 '9' picture symbols
  - 20 minus the number of decimal places, if the data item has 10 to 18 '9' picture symbols
- Internal floating-point data items are converted as if they were moved to a data item as follows:
  - For COMP-1: an external floating-point data item with PICTURE -9.9(8)E+99
  - For COMP-2: an external floating-point data item with PICTURE -9.9(17)E+99 (illegal because of the number of digit positions)
- Index data items are converted as if they were declared USAGE BINARY PICTURE S9(9).

After any conversion to character format, leading and trailing spaces and leading zeroes are eliminated, as described under “Trimming of generated XML data” on page 449.

If a data item after any conversion contains any characters that are illegal in XML content, as specified in the relevant XML specification, the original data value (that is, the value in the data item before any conversion or trimming) is represented in hexadecimal, and an element tag name with the prefix 'hex.' is substituted for the regular tag name. For example, if data item Customer-Name is found at run time to contain LOW-VALUES, the XML element tag name 'hex.Customer-Name' is used instead of the normal 'Customer-Name', and the content is represented as a string of pairs of zero digits.

Any remaining instances of the five characters & (ampersand), ' (apostrophe), > (greater-than sign), < (less-than sign), and “ (quotation mark) are converted into the equivalent XML references '&', ''', '>', '<', and '"', respectively.

Then, if *identifier-1* is a data item of category national, any nonnational values are converted to national format.

Trimming of generated XML data

Trimming is performed on data values after their conversion to character format. (Conversion is described under “Format conversion of elementary data” on page 448.)

For values converted from signed numeric values, the leading space is removed if the value is positive.

For values converted from numeric items, leading zeroes (after any initial minus sign) up to but not including the digit immediately before the actual or implied decimal point are eliminated. Trailing zeroes after a decimal point are retained. For example:

- -012.340 becomes -12.340.
- 0000.45 becomes 0.45.
- 0013 becomes 13.
- 0000 becomes 0.

Character values from data items of class alphabetic, alphanumeric, DBCS, and national have either trailing or leading spaces removed, depending on whether the corresponding data items have left (default) or right justification, respectively. That is, trailing spaces are removed from values whose corresponding data items do not specify the JUSTIFIED clause. Leading spaces are removed from values whose data items do specify the JUSTIFIED clause. If a character value consists solely of spaces, one space remains as the value after trimming is finished.

XML element name formation

In the XML documents that are generated from *identifier-2*, the XML element tag names are derived from the name of the data item specified by *identifier-2* and from any eligible data-names that are subordinate to *identifier-2* as follows:

- The exact mixed-case spelling of data-names from the data description entry is retained. The spellings from any references to that data item (for example, in an OCCURS DEPENDING ON clause) are not used.
- Data-names that start with a digit are prefixed by an underscore. For example, the data-name '3D' becomes XML tag name '_3D'.
- Data-names that start with the characters 'xml', in any combination of uppercase and lowercase, are prefixed by an underscore. For example, the data-name 'Xml' becomes XML tag name '_Xml'.
- Names of data items that are found at run time to contain characters that are illegal in XML version 1.0 content are prefixed by 'hex.', and the content itself is expressed in hexadecimal.

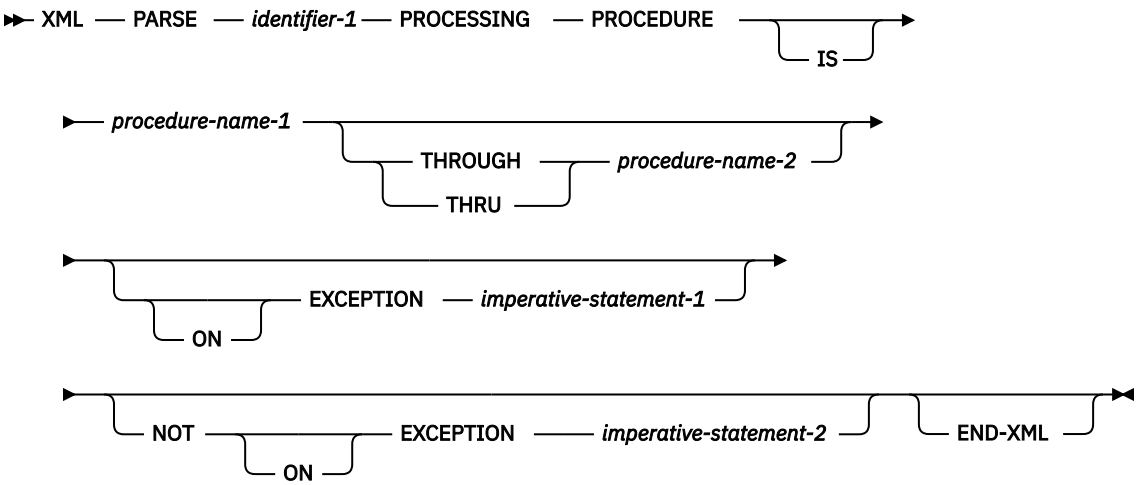
DBCS data-names, when translated to Unicode, must be legal as names in the XML specification, version 1.0.

For a discussion of the exception codes that special register XML-CODE can contain after execution of the XML GENERATE statement, see the *ILE COBOL Programming Guide*.

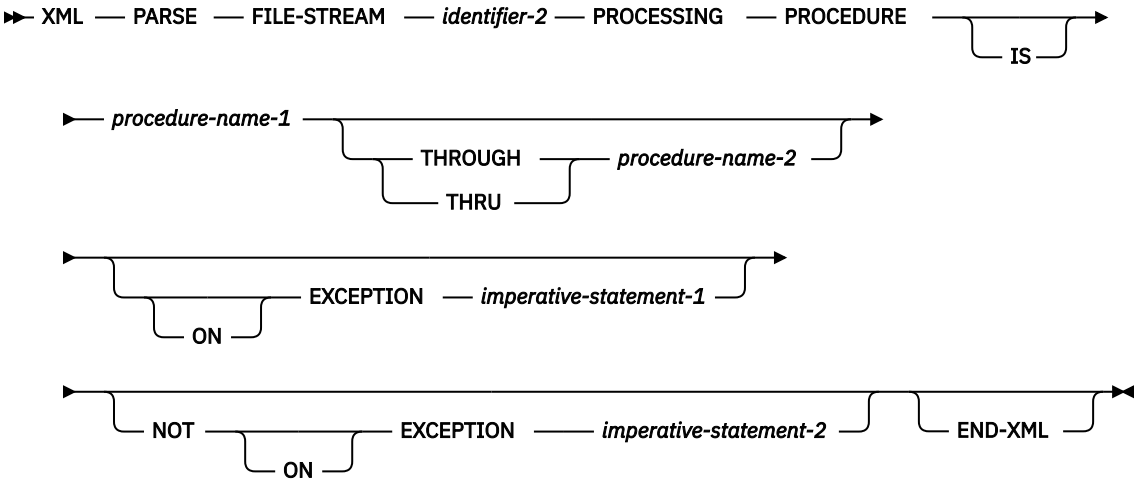
[IBM Extension] XML PARSE Statement

The XML PARSE statement is the ILE COBOL language interface to the high-speed XML parser that is part of the COBOL run time. The XML PARSE statement parses an XML document into its individual pieces and passes each piece, one at a time, to a user-written processing procedure.

XML Parse Statement – Format 1



XML Parse Statement – Format 2



identifier-1
Must be an alphanumeric or national data item that contains the XML document character stream. Identifier-1 cannot be a function-identifier.

If identifier-1 is alphanumeric, its contents must be encoded using one of the single-byte character sets listed under “Coded character sets for XML documents” on page 453. EBCDIC XML documents that do not contain an encoding declaration are parsed with the coded character set of the source member.

If identifier-1 is national, its contents must be encoded using the UCS-2 CCSID specified on the National CCSID compiler option or the NTLCCSID PROCESS option.

identifier-2
Identifier-2 must be an alphanumeric data item containing the absolute or relative path name of the stream file that contains the XML document. An absolute name starts with "/", for example "/u/user1/myxml". A relative path name does not start with "/", so this will be concatenated with the current directory. XML documents, including ASCII XML documents, located in the specified stream file that do not contain an encoding declaration are parsed with the coded character set of the stream file.

PROCESSING PROCEDURE phrase
Specifies the name of a procedure to handle the various events that the XML parser generates.

procedure-name-1
Specifies the first or only section or paragraph in the processing procedure.

procedure-name-2
Specifies the last section or paragraph in the processing procedure.

The processing procedure consists of the statements at which XML events are handled. The range of the processing procedure also includes all statements executed by CALL, EXIT, GO TO, GOBACK, and PERFORM statements in the range of the processing procedure.

The processing procedure must not *directly* execute an XML PARSE statement. However, if the processing procedure passes control to an outermost program by using a CALL statement, the target program can execute the same or a different XML PARSE statement. A program executing on multiple threads can execute the same XML statement or different XML statements simultaneously.

The compiler inserts a return mechanism after the last statement in the processing procedure. The processing procedure can terminate the run unit with a STOP RUN statement. It must not attempt to return to the parser with a GOBACK or EXIT PROGRAM statement.

For more details about the processing procedure, see “Control flow” on page 452 and “Processing procedures” on page 453.

ON EXCEPTION

The ON EXCEPTION phrase specifies imperative statements that are executed when the XML PARSE statement raises an exception condition.

An exception condition occurs when the XML parser detects an error in processing the XML document. The parser first signals an exception XML event by passing control to the processing procedure with special register XML-EVENT set to contain 'EXCEPTION'. The parser provides a numeric error code in special register XML-CODE, as detailed in the *ILE COBOL Programmer's Guide*.

An exception condition also occurs if the processing procedure deliberately terminates parsing by setting XML-CODE to -1 before returning to the parser from any normal XML event. In this case, the parser does not signal an EXCEPTION XML event.

If the ON EXCEPTION phrase is specified, the parser then transfers control to imperative-statement-1. If the ON EXCEPTION phrase is not specified, the NOT ON EXCEPTION phrase, if any, is ignored, and control is transferred to the end of the XML PARSE statement.

If the XML processing procedure handles the exception XML event and sets XML-CODE to zero before returning control to the parser, the exception condition no longer exists. If no other unhandled exceptions occur prior to the termination of the parser, control is transferred to imperative-statement-2 of the NOT ON EXCEPTION phrase, if specified.

NOT ON EXCEPTION

The NOT ON EXCEPTION phrase specifies imperative statements that are executed when no exception condition exists at the termination of XML PARSE processing.

If an exception condition does not exist at termination of XML PARSE processing, control is transferred to imperative-statement-2 of the NOT ON EXCEPTION phrase, if specified. If the NOT ON EXCEPTION phrase is not specified, control is transferred to the end of the XML PARSE statement. The ON EXCEPTION phrase, if specified, is ignored.

Special register XML-CODE contains zero after execution of the XML PARSE statement.

END-XML phrase

This explicit scope terminator serves to delimit the scope of the XML PARSE statement. END-XML permits a conditional XML PARSE statement to be nested in another conditional statement. END-XML can also be used with an XML PARSE statement that does not specify either the ON EXCEPTION or the NOT ON EXCEPTION phrase.

[End of IBM Extension]

Control flow

When the XML parser receives control from an XML PARSE statement, the parser analyzes the XML document and transfers control to procedure-name-1 at the following points in the process:

- The start of the parsing process
- When a document fragment is found
- When the parser detects an error in parsing the XML document
- The end of processing the XML document

Control returns to the XML parser when the end of the processing procedure is reached.

The exchange of control between the parser and the processing procedure continues until either:

- The entire XML document has been parsed, ending with the END-OF-DOCUMENT event.
- The parser detects an exception and the processing procedure does not reset special register XML-CODE to zero prior to returning to the parser.
- The processing procedure terminates parsing deliberately by setting XML-CODE to -1 prior to returning to the parser.

Then, the parser terminates and returns control to the XML PARSE statement with the XML-CODE special register containing the most recent value set by the parser or the processing procedure.

For each XML event passed to the processing procedure, the XML-CODE, XML-EVENT, and XML-TEXT or XML-NTEXT special registers contain information about the particular event. The content of the XML-CODE special register is defined during and after execution of an XML PARSE statement. The contents of all other XML special registers is undefined outside the range of the processing procedure.

For normal events, special register XML-CODE contains zero when the processing procedure receives control. For EXCEPTION events, XML-CODE contains one of the XML exception codes specified in the *ILE COBOL Programmer's Guide*. Special register XML-EVENT is set to the event name, such as 'START-OF-DOCUMENT'. Either XML-TEXT or XML-NTEXT contains the piece of the document corresponding with the event, as described in XML-EVENT.

For more information about the XML special registers, see Special registers.

For all kinds of XML events, if XML-CODE is not zero when the processing procedure returns control to the parser, the parser terminates without a further EXCEPTION event. Setting XML-CODE to -1 before returning to the parser from the processing procedure for an event other than EXCEPTION forces the parser to terminate with a user-initiated exception condition. For some EXCEPTION events, the processing procedure can set XML-CODE to zero to force the parser to continue, although subsequent results are unpredictable. When XML-CODE is zero, parsing continues until the entire XML document has been parsed or an unhandled exception condition occurs.

During parsing, the program that specified the XML PARSE statement must not be called recursively.

The XML PARSE statement must not be specified in a nested program, although the XML special registers may be referenced in nested programs.

For more information about the EXCEPTION event and exception processing, see the *ILE COBOL Programmer's Guide*.

Processing procedures

Keep in mind the following when coding your processing procedures:

- An XML processing procedure must not contain any EXIT PROGRAM or GOBACK statements.
- You can use ALTER, GO TO, and PERFORM statements in the processing procedure to transfer control to procedure-names outside the processing procedure. However, control must return to the processing procedure after a GO TO or PERFORM statement.
- A processing procedure can contain a CALL statement. The target program can contain an XML PARSE statement.

The *ILE COBOL Programmer's Guide* provides details on using the XML PARSE statement and processing procedures.

Coded character sets for XML documents

XML PARSE supports XML documents in national data items, in alphanumeric data items, and in IFS files with UCS-2 and single byte CCSIDs. Documents in national data items must be encoded using the Unicode UCS-2 CCSID specified on the National CCSID compiler option or the NTLCCSID PROCESS option. Documents in alphanumeric data items must be encoded using one of the explicitly supported single-byte EBCDIC CCSIDs shown in Supported EBCDIC CCSIDs for XML documents (Table 36 on page 453) or one of the ASCII CCSIDs shown in Supported ASCII CCSIDs for XML documents (Table 37 on page 454).

Table 36. ***Supported EBCDIC CCSIDs for XML documents***

CCSID	Description
1140, 37	USA, Canada, etc. Euro Country Extended CCSID (ECECP), Country Extended CCSID
1141, 273	Austria, Germany ECECP, CECP

Table 36. ***Supported EBCDIC CCSIDs for XML documents*** *(continued)*	
CCSID	**Description**
1142, 277	Denmark, Norway ECECP, CECP
1143, 278	Finland, Sweden ECECP, CECP
1144, 280	Italy ECECP, CECP
1145, 284	Spain, Latin America (Spanish) ECECP, CECP
1146, 285	UK ECECP, CECP
1147, 297	France ECECP, CECP
1148, 500	International ECECP, CECP
1149, 871	Iceland ECECP, CECP

Table 37. ***Supported ASCII CCSIDs for XML documents***	
CCSID	**Description**
813	ISO 8859-7 Greek / Latin
819	ISO 8859-1 Latin 1 / Open Systems
920	ISO 8859-9 Latin 5 (ECMA-128, Turkey TS-5881)

When you parse ASCII XML documents, the document fragments passed to the processing procedure in special register XML-TEXT are encoded in ASCII. Because ILE COBOL operations such as move and comparison rely on EBCDIC encoding or on national characters for proper operation, you must convert the document fragments before using them. To do this when the XML document is in a COBOL program, first convert from the ASCII CCSID of the XML document to national characters using the MOVE statement. Then, if necessary, convert the result from national characters to EBCDIC using the MOVE statement.

XML documents in a COBOL program encoded in other CCSIDs can be parsed by converting them to national characters using the MOVE statement. The individual pieces of document text passed to the processing procedure in special register XML-NTEXT can then be converted back to the original CCSID as necessary, using the MOVE statement.

When the XML document is in an IFS file, use the copy object (CPY) command to do the CCSID conversion. To make it easier to work with document fragments returned from the parser, it is recommended that you do the following before you use the document in an XML PARSE:

1. Characters preceding the '<' tag at the start of each xml record should be removed.
2. The end of each line in the IFS file must have only a CR (carriage return) and not a LF (line feed).
3. Convert XML documents to the UCS-2 CCSID specified on the National CCSID compiler option or the NTLCCSID PROCESS option, or convert XML documents to the CCSID of the job.
4. Manually change the encoding declaration in the XML document to specify the document's actual CCSID.

See the *ILE COBOL Programmer's Guide* for details on specifying the document encoding and how the parser determines encoding.

Special Registers

- XML-CODE
- XML-EVENT
- XML-NTEXT

- XML-TEXT

XML-CODE Special Register

The XML-CODE special register is used for the following purposes:

- To communicate status between the XML parser and the processing procedure that was identified in the XML PARSE statement
- To indicate either that an XML GENERATE statement executed successfully or that an exception occurred during XML generation

The XML parser sets XML-CODE prior to transferring control to the processing procedure for each event and at parser termination. You can reset XML-CODE prior to returning control from the processing procedure to the XML parser.

The XML-CODE special register has the implicit definition:

```
01 XML-CODE PICTURE S9(9) USAGE BINARY VALUE 0.
```

When used in nested programs, this special register is implicitly defined with the global attribute in the outermost program.

When the XML parser encounters an XML event, it sets XML-CODE and then passes control to the processing procedure. For all events except an EXCEPTION event, XML-CODE contains zero when the processing procedure receives control.

For an EXCEPTION event, the parser sets XML-CODE to an exception code indicating the nature of the exception. Exception codes are detailed in the *ILE COBOL Programmer's Guide*.

You can set XML-CODE before returning to the parser, as follows:

- To -1, after a normal event, to indicate that the parser is to terminate without causing an EXCEPTION event.
- To zero, after an EXCEPTION event for which continuation is allowed, to indicate that the parser is to continue processing. The parser will attempt to continue processing the XML document, but results are undefined.

If you set XML-CODE to any other value before returning to the parser, results are undefined.

When the parser returns control to the XML PARSE statement, XML-CODE contains the most recent value set either by the parser or by the processing procedure.

At termination of an XML GENERATE statement, XML-CODE contains either zero, indicating successful completion of XML generation, or a nonzero error code, indicating that an exception occurred during XML generation. XML GENERATE exception codes are detailed in the *ILE COBOL Programmer's Guide*.

XML-EVENT Special Register

The XML-EVENT special register is used to communicate event information from the XML parser to the processing procedure that was identified in the XML PARSE statement. Prior to passing control to the processing procedure, the XML parser sets the XML-EVENT special register to the name of the XML event, as described in Table 38 on page 456.

The XML-CODE special register has the implicit definition:

```
01 XML-EVENT USAGE DISPLAY PICTURE X(30) VALUE SPACE.
```

When used in nested programs, this special register is implicitly defined with the global attribute in the outermost program.

XML-EVENT cannot be used as a receiving data item.

Table 38. Contents of XML-EVENT and XML-TEXT or XML-NTEXT special registers

XML event (content of XML-EVENT)	Content of XML-TEXT or XML-NTEXT
ATTRIBUTE-CHARACTER	The single character corresponding with the predefined entity reference in the attribute value.
ATTRIBUTE-CHARACTERS	The value within quotes or apostrophes. This can be a substring of the attribute value if the value includes an entity reference.
ATTRIBUTE-NAME	The attribute name, the string to the left of =.
ATTRIBUTE-NATIONAL-CHARACTER	Regardless of the type of the XML document specified by identifier-1 in the XML PARSE statement, XML-TEXT is empty and XML-NTEXT contains the single national character corresponding with the (numeric) character reference.
COMMENT	The text of the comment between the opening character sequence "<!—" and the closing character sequence "—>".
CONTENT-CHARACTER	The single character corresponding with the predefined entity reference in the element content.
CONTENT-CHARACTERS	The element content between start and end tags. This can be a sub-string of the element content if the content contains an entity reference or another element.
CONTENT-NATIONAL-CHARACTER	Regardless of the type of the XML document specified by identifier-1 in the XML PARSE statement, XML-TEXT is empty and XML-NTEXT contains the single national character corresponding with the (numeric) character reference.
DOCUMENT-TYPE-DECLARATION	The entire document type declaration including the opening and closing character sequences, "<! DOCTYPE" and ">".
ENCODING-DECLARATION	The value, between quotes or apostrophes, of the encoding declaration in the XML declaration.
END-OF-CDATA-SECTION	Always contains the string "]]>".
END-OF-DOCUMENT	Null, zero-length.
END-OF-ELEMENT	The name of the end element tag or empty element tag.
EXCEPTION	The part of the document successfully scanned, up to and including the point at which the exception was detected. [1]
PROCESSING-INSTRUCTION-DATA	The rest of the processing instruction, not including the closing sequence, "?>", but including trailing, and not leading, white space characters.
PROCESSING-INSTRUCTION-TARGET	The processing instruction target name, which occurs immediately after the processing instruction opening sequence, "<?".

Table 38. Contents of XML-EVENT and XML-TEXT or XML-NTEXT special registers (continued)	
XML event (content of XML-EVENT)	**Content of XML-TEXT or XML-NTEXT**
STANDALONE-DECLARATION	The value, between quotes or apostrophes, of the standalone declaration in the XML declaration.
START-OF-CDATA-SECTION	Always contains the string "<![CDATA[".
START-OF-DOCUMENT	The entire document.
START-OF-ELEMENT	The name of the start element tag or empty element tag, also known as the element type.
UNKNOWN-REFERENCE-IN-CONTENT	The entity reference name, not including the "&" and ";" delimiters.
UNKNOWN-REFERENCE-IN-ATTRIBUTE	The entity reference name, not including the "&" and ";" delimiters.
VERSION-INFORMATION	The value, between quotes or apostrophes, of the version declaration in the XML declaration. This is currently always "1.0".

Note:

1. Exceptions for encoding conflicts are signaled before parsing begins. For these exceptions, XML-TEXT is either zero-length or contains just the encoding declaration value from the document. See the *ILE COBOL Programmer's Guide* for information on XML exception codes.

XML-NTEXT Special Register

The XML-NTEXT special register is defined during XML parsing to contain document fragments that are USAGE NATIONAL.

XML-NTEXT is an elementary national data item of the length of the contained XML document fragment. The length of XML-NTEXT can vary from zero through 8,000,000 national character positions. The maximum byte length is 16,000,000.

Note: There is no equivalent COBOL data description entry.

When used in nested programs, this special register is implicitly defined with the global attribute in the outermost program.

The parser sets XML-NTEXT to the document fragment associated with an event before transferring control to the processing procedure, in these cases:

- When the operand of the XML PARSE statement is a national data item
- For the ATTRIBUTE-NATIONAL-CHARACTER event, and
- For the CONTENT-NATIONAL-CHARACTER event.

When **XML-NTEXT** is set, the **XML-TEXT** special register has a length of zero. At any given time, only one of the two special registers XML-NTEXT and XML-TEXT has a non-zero length.

Use the LENGTH function to determine the number of national characters that XML-NTEXT contains. The LENGTH OF special register for XML-NTEXT has the number of bytes, rather than the number of national characters, contained in XML-NTEXT.

Note: The START-DOCUMENT event may be greater than 8,000,000 national characters. In this case, special register XML-NTEXT will contain only the first 8,000,000 characters of the event, and its LENGTH will be set to 16,000,000 bytes.

XML-NTEXT cannot be used as a receiving item.

XML-TEXT Special Register

The XML-TEXT special register is defined during XML parsing to contain document fragments that are of class alphanumeric.

XML-TEXT is an elementary alphanumeric data item of the length of the contained XML document fragment. The length of XML-TEXT can vary from zero through 16,000,000 bytes.

Note: There is no equivalent COBOL data description entry.

When used in nested programs, this special register is implicitly defined with the global attribute in the outermost program.

The parser sets XML-TEXT to the document fragment associated with an event before transferring control to the processing procedure when the operand of the XML PARSE statement is an alphanumeric data item, except for the ATTRIBUTE-NATIONAL-CHARACTER event and the CONTENT-NATIONAL-CHARACTER event.

When **XML-TEXT** is set, the **XML-NTEXT** special register has a length of zero. At any given time, only one of the two special registers XML-NTEXT and XML-TEXT has a non-zero length.

Use the LENGTH function or the LENGTH OF special register for XML-TEXT to determine the number of bytes that XML-TEXT contains.

Note: The START-DOCUMENT event may be greater than 16,000,000 characters. In this case, special register XML-TEXT will contain only the first 16,000,000 characters of the event, and its LENGTH will be set to 16,000,000 bytes.

XML-TEXT cannot be used as a receiving item.

Intrinsic Functions

Data processing problems often require the use of values that are not directly accessible in the data storage associated with the object program, but instead must be derived through performing operations on other data. An intrinsic function is a function that performs a mathematical, character, or logical operation, and thereby allows you to make reference to a data item whose value is derived automatically during the execution of the object program.

The functions can be grouped into six categories, based on the type of service performed: mathematical, statistical, date/time, financial, character-handling, and general.

You can reference a function by specifying its name, along with any required arguments, in a Procedure Division statement.

Functions are elementary data items, and return alphanumeric, DBCS, numeric, integer, boolean, or date-time values. Functions cannot serve as receiving operands.

Function Definition and Evaluation

The class and characteristics of a function, and the number and types of arguments it requires, are determined by its function definition. These characteristics include:

- For some functions, the class and characteristics are determined by the arguments to the function
- For alphanumeric functions, the size of the returned value

[IBM Extension]

- For DBCS functions, the size of the returned value
- For date-time functions, the length of the returned value
- For numeric and integer functions, the sign of the returned value, and whether the function is integer
- The actual value returned by the function.

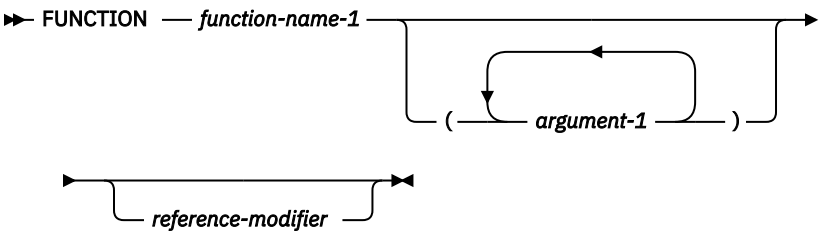
[End of IBM Extension]

The evaluation of any intrinsic function is not affected by the context in which it appears; in other words, function evaluation is not affected by operations or operands outside the function. However, evaluation of a function can be affected by the attributes of its arguments.

Specifying a Function

The general format of a function-identifier is:

Format



function-name-1
Function-name-1 must be one of the Intrinsic Function names.

argument-1
Argument-1 must be an identifier, literal (other than a figurative constant), or arithmetic expression.

reference-modifier
Can be specified only for functions of the category alphanumeric or DBCS.

The following examples show an intrinsic function invocation for an alphanumeric source statement and a numeric source statement.

The alphanumeric source statement:

```
MOVE FUNCTION UPPER-CASE("hello") TO DATA-NAME.
```

replaces each lowercase letter in the argument with the corresponding uppercase letter, resulting in the movement of HELL0 into DATA-NAME.

The numeric source statement,

```
COMPUTE NUM-ITEM = FUNCTION MEAN(A B C)
```

Adds the values of A, B, and C then divides by 3, and places the result in NUM-ITEM.

Within a Procedure Division statement, each function-identifier is evaluated at the same time as any reference modification or subscripting associated with an identifier in that same position would be evaluated.

Types of Functions

There are seven types of functions:

- Alphanumeric
- Numeric

[IBM Extension]

- DBCS
- National
- Date-Time
- Boolean

[End of IBM Extension]

- Integer

Alphanumeric functions are of the class and category alphanumeric. The value returned has an implicit usage of DISPLAY and is in standard data format characters. The number of character positions in the value returned is determined by the function definition.

Numeric functions are of the class and category numeric. The returned value is always considered to have an operational sign and is a numeric intermediate result. For more information, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

[IBM Extension]

DBCS functions are of the class and category DBCS. The value returned has an implicit usage of DISPLAY-1. The number of character positions in the value returned is determined by the function definition.

National functions are of the class and category NATIONAL. The value returned has an implicit usage of NATIONAL. The number of character positions in the value returned is determined by the function definition.

Date-Time functions are of the class date-time and category date, time, or timestamp. The value returned has an implicit usage of DISPLAY. The number of character positions in the value returned is determined by the function definition.

Boolean functions are of class and category boolean. The value returned has an implicit usage of DISPLAY, and is either a boolean true (B"1") or a boolean false (B"0").

[End of IBM Extension]

Integer functions are of the class and category numeric. The returned value is always considered to have an operational sign and is an integer intermediate result. The number of digit positions in the value returned is determined by the function definition. For more information, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

Rules for Usage

Alphanumeric Functions

An alphanumeric function can be specified wherever an identifier is permitted in the general formats, and where the rules associated with the general formats do not specifically prohibit reference to functions. However it cannot be specified:

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

A reference modification for an alphanumeric function is allowed. If reference modification is specified for a function, the evaluation of the reference modification takes place immediately after the evaluation of the function.

An alphanumeric function can be referenced as an argument for a function which allows an alphanumeric argument.

Numeric Functions

A numeric function can be used only where an arithmetic expression can be specified.

A numeric function can be referenced as an argument for a function that allows a numeric argument.

A numeric function cannot be used where an integer operand is required, even if the particular reference will yield an integer value. The INTEGER or INTEGER-PART functions can be used to force the type of a numeric argument to be an integer.

[IBM Extension]

DBCS Functions

A DBCS function can be specified wherever a DBCS identifier is permitted in the general formats, and wherever the rules associated with the general formats do not specifically prohibit reference to functions. However, it cannot be specified:

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

A reference modification for a DBCS function is allowed. If reference modification is specified for a function, the evaluation of the reference modification takes place immediately after the evaluation of the function.

A DBCS function can be referenced as an argument for a function that allows a DBCS argument.

National Functions

A national function can be specified wherever a national identifier is permitted in the general formats, and wherever the rules associated with the general formats do not specifically prohibit reference to functions. However, it cannot be specified:

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

A national function can be referenced as an argument for a function that allows a national argument.

Date-Time Functions

A date-time function can be specified wherever a date-time identifier is permitted in the general formats, and wherever the rules associated with the general formats do not specifically prohibit reference to functions. However, it cannot be specified:

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

A date-time function is allowed as part of a relation condition. If a date-time function is specified in a relation condition, the evaluation of the relation condition takes place immediately after the evaluation of the function.

A date-time function can be referenced as an argument for a function that allows a date-time argument.

Boolean Functions

A boolean function can be specified wherever a boolean identifier is permitted in the general formats, and wherever the rules associated with the general formats do not specifically prohibit reference to functions. However, it cannot be specified:

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

A boolean function is allowed as part of a relation condition. If a boolean function is specified in a relation condition, the evaluation of the relation condition takes place immediately after the evaluation of the function.

A boolean function can be referenced as an argument for a function that allows a boolean argument.

[End of IBM Extension]

Integer Functions
An integer function can be used only where an arithmetic expression can be specified.

An integer function can be referenced as an argument for a function that allows an integer argument.

Special Usage Notes:
Identifiers in the USING phrase of the CALL statement must not be a function-identifier.

The COPY statement allows function-identifiers of all types in the REPLACING phrase.

Arguments

The values returned by some functions are determined by the arguments specified in the function-identifier when the functions are evaluated. Some functions require no arguments; others require a fixed number of arguments; and still others allow a variable number of arguments.

An argument must be one of the following:

- An identifier
- An arithmetic expression
- A function-identifier
- A literal other than a figurative constant
- A special-register
- A mnemonic-name
- A keyword.

The argument to a function can be any function or expression containing a function, including another evaluation of the same function, whose result meets the category requirement for the argument.

See “Function Definitions” on page 465 for function specific argument specifications.

The types of arguments are:

Alphabetic
An elementary data item of the class alphabetic or a nonnumeric literal containing only alphabetic characters. The content of the argument is used to determine the value of the function. The length of the argument can be used to determine the value of the function.

Alphanumeric
A data item of the class alphabetic or alphanumeric or a nonnumeric literal. The content of the argument will be used to determine the value of the function. The length of the argument can be used to determine the value of the function.

[IBM Extension]

Boolean
A data item of class boolean, or a boolean literal.

DBCS
A data item of the class DBCS or a DBCS literal. The content of the argument is used to determine the value of the function. The length of the argument can be used to determine the value of the function.

National
A data item of the class NATIONAL or a national literal. The content of the argument is used to determine the value of the function. The length of the argument can be used to determine the value of the function.

Date-Time
An data item of the class date-time. The content of the argument is used to determine the value of the function. The length of the argument can be used to determine the value of the function.

[End of IBM Extension]

Index
An index data item. The size associated with the argument may be used in determining the value of the argument.

Integer
An arithmetic expression that always results in an integer value. The value of this expression, including its sign, is used to determine the value of the function.

Numeric
An arithmetic expression, whose value, including its sign, is used to determine the value of the function.

[IBM Extension]

Keyword
A keyword should be specified in accordance with the function definition.

Mnemonic-Name
A mnemonic-name defined in the SPECIAL-NAMES paragraph shall be specified. The feature associated with the mnemonic-name may be used in determining the value of the function.

Pointer
A pointer identifier. The size associated with the argument may be used in determining the value of the function.

Type Declaration
A type-name shall be specified. The size associated with the type declaration may be used in determining the value of the function.

Special-Register
A special-register should be specified in accordance with the function definition. The information associated with the special-register may be used in determining the value of the function.

[End of IBM Extension]

Some functions place constraints on their arguments, such as the range of values acceptable. If the values assigned as arguments for a function do not comply with specified constraints, the returned value is undefined.

If a nested function is used as an argument, the evaluation of its arguments will not be affected by the arguments in the outer function.

Only those arguments at the same function level interact with each other. This interaction occurs in two areas:

- The computation of an arithmetic expression that appears as a function argument will be affected by other arguments for that function.
- The evaluation of the function takes into consideration the attributes of all of its arguments.

[IBM Extension]

Floating-point literals are allowed wherever a numeric argument is allowed, and in arithmetic expressions that are used in functions that allow a numeric argument. They are *not* allowed where an integer argument is required.

External floating-point items are allowed wherever a numeric argument is allowed, and in arithmetic expressions that are used in functions that allow a numeric argument.

External floating-point items are *not* allowed where an integer argument is required, or where an argument of alphanumeric class is allowed in a function identification, such as in the LOWER-CASE, REVERSE, UPPER-CASE, NUMVAL, and NUMVAL-C functions.

[End of IBM Extension]

Order of Precedence for the Evaluation of Function Arguments

When a function is evaluated, its arguments are evaluated individually in the order specified in the list of arguments, from left to right. The argument being evaluated can be a function-identifier, or it can be an expression containing function-identifiers.

If an arithmetic expression is specified as an argument, and if the first operator in the expression is a unary plus or a unary minus, it must be immediately preceded by a left parenthesis. For example, function `MEAN(x-y z)` would be the mean of two arguments: `x-y` and `z`.

To get the mean of the unary minus of `y`, the parentheses would be added as follows:

```
MEAN((x) (-y) z)
```

Note: As in the preceding example, when you are taking the mean of unary minus and the unary minus is not the first of multiple arguments, you also need to enclose the preceding arguments in brackets. This ensures that the unary minus, (-y) in this example, will not be interpreted as a subscript.

ALL Subscripting

When a function allows an argument to be repeated a variable number of times, you can refer to a table by specifying the data-name and any qualifiers that identify the table. This can be followed immediately by subscripting where one or more of the subscripts is the word ALL.

Note: The evaluation of an ALL subscript must result in at least one argument or the value returned by the function will be undefined; however, the situation can be diagnosed at run time by specifying the *RANGE compiler option.

Specifying ALL as a subscript is equivalent to specifying all table elements possible using every valid subscript in that subscript position.

For a table argument specified as "Table-name(ALL)", the order of the implicit specification of each table element as an argument is from left to right, where the first (or leftmost) argument is "Table-name(1)" and ALL has been replaced by 1. The next argument is "Table-name(2)", where the subscript has been incremented by 1. This process continues, with the subscript being incremented by 1 to produce an implicit argument, until the ALL subscript has been incremented through its range of values.

For example,

```
FUNCTION MEAN(Table(ALL))
```

is equivalent to

```
FUNCTION MEAN(Table(1) Table(2) Table(3)... Table(n))
```

where n is the number of elements in Table.

If there are multiple ALL subscripts, "Table-name(ALL, ALL, ALL)", the first implicit argument is "Table-name(1, 1, 1)", where each ALL has been replaced by 1. The next argument is "Table-name(1, 1, 2)", where the rightmost subscript has been incremented by 1. The subscript represented by the rightmost ALL is incremented through its range of values to produce an implicit argument for each value.

Once a subscript specified as ALL has been incremented through its range of values, the next subscript to the left that is specified as ALL is incremented by 1. Each subscript specified as ALL to the right of the newly incremented subscript is set to 1 to produce an implicit argument. Once again, the subscript represented by the rightmost ALL is incremented through its range of values to produce an implicit argument for each value. This process is repeated until each subscript specified as ALL has been incremented through its range of values.

For example,

```
FUNCTION MEAN(Table(ALL, ALL))
```

is equivalent to

```
FUNCTION MEAN(Table(1, 1) Table(1, 2) Table(1, 3)... Table(1, n)
              Table(2, 1) Table(2, 2) Table(2, 3)... Table(2, n)
              Table(3, 1) Table(3, 2) Table(3, 3)... Table(3, n)
                                      .
                                      .
                                      .
              Table(m, 1) Table(m, 2) Table(m, 3)... Table(m, n))
```

where n is the number of elements in the column dimension of Table, and m is the number of elements in the row dimension of Table.

ALL subscripts can be combined with literal, data-name, or index-name subscripts to reference multidimensional tables.

For example,

```
FUNCTION MEAN(Table(ALL, 2))
```

is equivalent to

```
FUNCTION MEAN(Table(1, 2)
              Table(2, 2)
              Table(3, 2)
                   .
                   .
                   .
              Table(m, 2))
```

where m is the number of elements in the row dimension of Table.

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

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

If the ALL subscript is associated with an OCCURS DEPENDING ON clause, the range of values is determined by the object of the OCCURS DEPENDING ON clause.

For example, given a payroll record definition such as:

```
01 PAYROLL.
  02 PAYROLL-WEEK   PIC 99.
  02 PAYROLL-HOURS  PIC 999 OCCURS 1 TO 52
     DEPENDING ON PAYROLL-WEEK.
```

The following COMPUTE statement could be used to identify the mean hours worked in any week:

The following COMPUTE statements could be used to identify total year-to-date hours, the maximum hours worked in any week, and the specific week corresponding to the maximum hours:

```
     COMPUTE YTD-HOURS = FUNCTION SUM (PAYROLL-HOURS(ALL))
     COMPUTE MAX-HOURS = FUNCTION MAX (PAYROLL-HOURS(ALL))
     COMPUTE MAX-WEEK  = FUNCTION ORD-MAX (PAYROLL-DAYS(ALL))
```

In this function invocation the subscript ALL is used to reference all elements of the PAYROLL-HOURS array (depending on the execution time value of the PAYROLL-WEEK field).

Function Definitions

Table 39 on page 466 provides an overview of the argument type, function type and value returned for each of the intrinsic functions. Argument types and function types are abbreviated as follows:

- A = alphabetic

[IBM Extension]

- B = boolean
- D = DBCS

- DA = date-time

[End of IBM Extension]

- I = integer
- IX = Index

[IBM Extension]

- K = keyword
- M = mnemonic-name

[End of IBM Extension]

- N = numeric
- NL = national

[IBM Extension]

- P = pointer
- S = special-register
- T = type-name

[End of IBM Extension]

- X = alphanumeric
- U = national (for Universal Character Set and Unicode)

Note: The number associated with these abbreviations, identifies which function argument is being referred to.

Table 39. Table of Functions

FUNCTION-NAME	ARGUMENT TYPES	FUNCTION TYPES	VALUE RETURNED
ACOS	N1	N	Arccosine of N1
ADD-DURATION[1]	DA1,K2, I3	DA	A date-time item with the duration added.
ANNUITY	N1, I2	N	Approximates an annuity paid at an interest rate of N1 for I2 periods.
ASIN	N1	N	Arcsine of N1
ATAN	N1	N	Arctangent of N1
CHAR	I1	X	Character in position I1 of program collating sequence
CONVERT-DATE-TIME[1]	DA1 or X1 or I1, K2, X3 or K3 or S3, M4 or S4	DA	Converted date-time item
COS	N1	N	Cosine of N1
CURRENT-DATE	None	X	Current date and time and difference from Greenwich Mean Time
DATE-OF-INTEGER	I1	I	Standard date equivalent (YYYYMMDD) of integer date

Table 39. Table of Functions (continued)

FUNCTION-NAME	ARGUMENT TYPES	FUNCTION TYPES	VALUE RETURNED
DATE-TO-YYYYMMDD[1]	I1 or I1, I2	I	Converts year in standard date from 2 digits to 4 digits
DAY-OF-INTEGER	I1	I	Julian date equivalent (YYYYDDD) of integer date
DAY-TO-YYYYDDD[1]	I1 or I1, I2	I	Converts year in Julian date from 2 digits to 4 digits
DISPLAY-OF	NL1	X	Each character in NL1 converted to a corresponding character representation using a code page identified by I2, if specified, or a default code page selected at compile time if I2 is unspecified
	NL1, I2		
	NL1, X2 or D2		
EXTRACT-DATE-TIME[1]	DA1, X2 or K2	I or X	Part of an extracted date, time, or timestamp item
FACTORIAL	I1	I	Factorial of I1
FIND-DURATION[1]	DA1, DA2, K3	I	The integer duration between 2 date-time items
INTEGER	N1	I	The greatest integer not greater than N1
INTEGER-OF-DATE	I1	I	Integer date equivalent of standard date (YYYYMMDD)
INTEGER-OF-DAY	I1	I	Integer date equivalent of Julian date (YYYYDDD)
INTEGER-PART	N1	I	Integer part of argument
LENGTH	A1, N1, X1, D1, B1, T1, DA1, P1, IX1, NL	I	Length of argument
LOCALE-DATE[1]	X1 or D1, M2	X	Date string formatted according to locale specified
LOCALE-TIME[1]	X1 or D1, M2	X	Time string formatted according to locale specified
LOG	N1	N	Natural logarithm of N1
LOG10	N1	N	Logarithm to base 10 of N1
LOWER-CASE	A1 or X1	X	All letters in the argument are set to lowercase
	D1	D	
	NL	NL	

Table 39. Table of Functions (continued)

FUNCTION-NAME	ARGUMENT TYPES	FUNCTION TYPES	VALUE RETURNED
MAX	A1...	X	Value of maximum argument; note that the type of function depends on the arguments
	I1...	I	
	N1...	N	
	X1...	X	
	U1...	U	
MEAN	N1...	N	Arithmetic mean of arguments
MEDIAN	N1...	N	Median of arguments
MIDRANGE	N1...	N	Mean of maximum and minimum arguments
MIN	A1...or	X	Value of minimum argument; note that the type of function depends on the arguments
	I1...or	I	
	N1...or	N	
	X1...or	X	
	U1...	U	
MOD	I1, I2	I	I1 modulo I2
NATIONAL-OF	A1 or X1 or D1	NL	The characters in argument-1 converted to national characters, using the code page identified by I2, if specified, or a default code page selected at compile time if I2 is unspecified
	A1 or X1 or D1, I2		
	A1 or X1 or D1, NL2		
NUMVAL	X1	N	Numeric value of simple numeric string
NUMVAL-C	X1 or X1,X2	N	Numeric value of numeric string with optional commas and currency sign
ORD	A1 or X1	I	Ordinal position of the argument in collating sequence
ORD-MAX	A1..., N1...,	I	Ordinal position of maximum argument
	X1..., or U1...		
ORD-MIN	A1..., N1...,	I	Ordinal position of minimum argument
	X1..., or U1...		
PRESENT-VALUE	N1, N2...	N	Present value of a series of future period-end amounts, N2, at a discount rate of N1
RANDOM	I1 or none	N	Random number
RANGE	I1...	I	Value of maximum argument minus value of minimum argument; note that the type of function depends on the arguments
	N1...	N	

Table 39. Table of Functions (continued)

FUNCTION-NAME	ARGUMENT TYPES	FUNCTION TYPES	VALUE RETURNED
REM	N1, N2 (N2 is nonzero)	N	Remainder of N1 divided by N2
REVERSE	A1 or X1	X	Reverse order of the characters of the argument
	D1	D	
	NL	NL	
SIN	N1	N	Sine of N1
SQRT	N1	N	Square root of N1
STANDARD DEVIATION	N1...	N	Standard deviation of arguments
SUM	I1...	I	Sum of arguments; note that the type of function depends on the arguments
	N1...	N	
SUBTRACT-DURATION[1]	DA1, K2, I3	DA	A date-time item with the duration subtracted
TAN	N1	N	Tangent of N1
TEST-DATE-TIME[1]	DA1 or X1 or I1, K2, X3 or K3 or S3, M4 or S4	B	True (B"1") if valid date-time item, otherwise false
TRIM[1]	A1 or X1	X	String with left and right blanks or specified characters trimmed
	A1, A2 or X1, X2		
	D1 or D1, D2	D	
	NL1 or NL1, NL2	NL	
TRIML[1]	A1 or X1	X	String with left blanks or specified characters trimmed
	A1, A2 or X1, X2		
	D1 or D1, D2	D	
	NL1 or NL1, NL2	NL	
TRIMR[1]	A1 or X1	X	String with right blanks or specified characters trimmed
	A1, A2 or X1, X2		
	D1 or D1, D2	D	
	NL1 or NL1, NL2	NL	
UPPER-CASE	A1 or X1	X	All letters in the argument are set to uppercase
	D1	D	
	NL	NL	
UTF8STRING[1]	A1, X1, D1 or NL1	A	Variable length UTF-8 string
VARIANCE	N1...	N	Variance of arguments
WHEN-COMPILED	None	X	Date and time when program was compiled

Table 39. Table of Functions (continued)			
FUNCTION-NAME	**ARGUMENT TYPES**	**FUNCTION TYPES**	**VALUE RETURNED**
YEAR-TO-YYYY[1]	I1 or I1, I2	I	Converts year from 2 digits to 4 digits
Note: [1] IBM Extension			

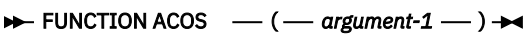
The following sections contain the function definitions for each of the Intrinsic Functions summarized in Table 39 on page 466.

ACOS

The ACOS function returns a numeric value in radians that approximates the arccosine of the argument specified.

The function type is numeric.

Format



argument-1

Must be class numeric. The value of argument-1 must be greater than or equal to -1 and less than or equal to +1.

The returned value is the approximation of the arccosine of the argument, and is greater than or equal to zero and less than or equal to Pi.

[IBM Extension] ADD-DURATION

The ADD-DURATION function adds a duration to a date, time, or timestamp item, and returns the modified item.

The function type is date-time.

The length of the returned value depends on the length of the date, time, or timestamp item specified in argument-1. The returned value will be truncated to the length of argument-1.

If a duration is added to a date item, the date returned must fall within a certain range:

- For 4-digit dates, the range must be 0001/01/01 through 9999/12/31
- For 2-digit dates, the range must be 0001/01/01 through 9999/12/31, but the year is truncated to 2 digits
- For a 3-digit year (a 1-digit century and a 2-digit year), the range must be 1900/01/01 through 2899/12/31 (the default). This range can be changed by specifying the DATTIM PROCESS statement option.

If a duration is added to a 2-digit date item, the range is the same as for a 4-digit year, but the year in the value returned is truncated to 2 digits.

Format

argument-1

Must be date, time, or timestamp data item.

Argument-1 is a data item containing a value to which a duration is added. The duration is specified in argument-2 and argument-3.

argument-2
Argument-2 is a keyword that represents a duration. The valid duration keywords are:

- YEARS
- MONTHS
- DAYS
- HOURS
- MINUTES
- SECONDS
- MICROSECONDS
- PICOSECONDS

The duration keyword must be consistent with argument-1. For example, the duration keywords most obey the following rules:

1. YEARS, MONTHS, and DAYS can only be added to a date or timestamp item.
2. HOURS, MINUTES, SECONDS, and MICROSECONDS can only be added to a time or timestamp item.
3. PICOSECONDS can only be added to a timestamp item.

argument-3
Must be an integer arithmetic expression. Argument-3 is the number of units of the duration, as specified in argument-2, that are to be added to argument-1.

Argument-3 can be a negative integer, but the function only takes its absolute value. If argument-3 is longer than 9 digits, it is truncated.

Argument-2 and argument-3 can be repeated. There should be no duplicate argument-2 in one intrinsic function.

If a duration is added to a date, and the result is invalid, the date is adjusted. For example, if a duration of 1 month is added to the date `March 31, 1997`, the result would be the invalid date `April 31, 1997`. This date would be adjusted to the valid date `April 30, 1997`.

[End of IBM Extension]

Examples

The following examples show how the ADD-DURATION intrinsic function can be used:

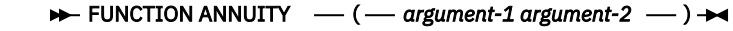
```
MOVE FUNCTION ADD-DURATION (date-3 MONTHS 1)
     TO  date-2.
MOVE FUNCTION ADD-DURATION (date-3 MONTHS int-1 * 2)
     TO  date-1.
MOVE FUNCTION ADD-DURATION (date-1  YEARS 1 MONTHS 5 DAYS 23)
     TO date-2.
```

ANNUITY

The ANNUITY function returns a numeric value that approximates the ratio of an annuity paid at the end of each period, for a given number of periods, at a given interest rate, to an initial value of one. The number of periods is specified by argument-2; the rate of interest is specified by argument-1. For example, if argument-1 is zero and argument-2 is four, the value returned is the approximation of the ratio 1 / 4.

The function type is numeric.

Format



argument-1
Must be class numeric. The value of argument-1 must be greater than or equal to zero.

argument-2
Must be a positive integer.

When the value of argument-1 is zero, the value returned by the function is the approximation of:

```
1 / ARGUMENT-2
```

When the value of argument-1 is not zero, the value of the function is the approximation of:

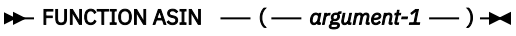
```
ARGUMENT-1 / (1 - (1 + ARGUMENT-1) ** (- ARGUMENT-2))
```

ASIN

The ASIN function returns a numeric value in radians that approximates the arcsine of the argument specified.

The function type is numeric.

Format



argument-1
Must be class numeric. The value of argument-1 must be greater than or equal to -1 and less than or equal to +1.

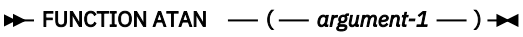
The returned value is the approximation of the arcsine of argument-1, and is greater than or equal to -Pi/2 and less than or equal to +Pi/2.

ATAN

The ATAN function returns a numeric value in radians that approximates the arctangent of the argument specified.

The function type is numeric.

Format



argument-1
Must be class numeric.

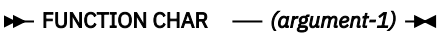
The returned value is the approximation of the arctangent of argument-1, and is greater than -Pi/2 and less than +Pi/2.

CHAR

The CHAR function returns a one-character alphanumeric value that is a character in the program collating sequence having the ordinal position equal to the value of the argument specified.

The function type is alphanumeric.

Format



argument-1
Must be an integer. The value must be greater than zero and less than or equal to the number of positions in the collating sequence.

If more than one character has the same position in the program collating sequence, the character returned as the function value is that of the first literal specified for that character position in the ALPHABET clause.

If the current program collating sequence was not specified by an ALPHABET clause, the EBCDIC collating sequence is used.

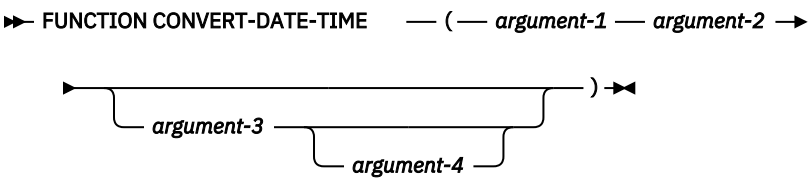
[IBM Extension] CONVERT-DATE-TIME

The CONVERT-DATE-TIME function takes an item of class alphanumeric, numeric, or date-time and returns a date-time item.

The function type is date-time.

The length of the returned value depends on the length allowed for the format of the date, time, or timestamp item specified in argument-2 through argument-4.

Format



argument-1
Can be:

- A date, time, or timestamp item
- An item of class alphanumeric
- A non-numeric literal
- An item of class numeric integer.

argument-2
Specifies the category of the return value and must be one of the following keywords:

- DATE
- TIME
- TIMESTAMP.

If argument-1 is a date, time, or timestamp item, CONVERT-DATE-TIME can only convert:

- A date to a date, or a timestamp
- A time to a time, or a timestamp
- A timestamp to a date, a time, or a timestamp.

If argument-2 is TIMESTAMP, argument-3 can only be specified with FORMAT OF special register, and argument-4 cannot be specified.

If argument-1 is a date-time item, a date-time move is done.

If argument-1 is a numeric integer, the returned date-time item will be right-justified and truncated, if it is longer than what is allowed by the date-time format specified in argument-3.

If argument-1 is anything else, the returned date-time item will be left-justified and truncated, if it is longer than what is allowed by the date-time format specified in argument-3.

argument-3
Specifies the format of a date or time item. It must be:

- A nonnumeric literal at least 2 characters long
- The keyword LOCALE

- The FORMAT OF special register.

For a list of valid literals and the rules that this argument must follow, refer to the SPECIAL-NAMES FORMAT clause described in "FORMAT Clause" on page 89.

Argument-3 should represent a category that is referred to by argument-2.

If argument-3 is the keyword LOCALE, then the format of the date or time is based on a LOCALE. If argument-4 is not specified, the current locale is used, otherwise the locale associated with the mnemonic-name or the LOCALE OF special register is used.

If argument-3 is not specified, the format of the returned value is dependent on the SPECIAL-NAMES FORMAT clause. If no format has been defined in the SPECIAL-NAMES paragraph, *ISO format is used. For TIMESTAMP, if argument-3 is not specified, the default format of @Y-%m-%d-%H%M %S.@Sm is used.

argument-4

Must be a mnemonic-name associated with a LOCALE, or the LOCALE OF special register.

Argument-4 must follow these rules:

- If argument-4 is specified and argument-3 is a locale-based format literal, for example contains %p, then the locale-based format literal would use the locale specified in argument-4 to determine the actual value of the conversion specifiers.
- If argument-3 is a locale-based format literal (for example, contains %p) and argument-4 is not specified, the locale-based format literal would use the current locale to determine the actual value of the conversion specifiers.
- If argument-3 is a locale-based format literal (for example, contains %p), and the LOCALE OF special register is used to refer to a non-locale item, the locale-based format literal would use the default locale to determine the actual value of the conversion specifiers.

[End of IBM Extension]

Examples

The following examples show how the CONVERT-DATE-TIME intrinsic function can be used:

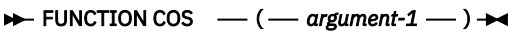
```
MOVE FUNCTION CONVERT-DATE-TIME ('95/05/30' DATE)
     TO  date-1.
MOVE FUNCTION CONVERT-DATE-TIME
     ('95/05/30' DATE '%y/%m/%d')
     TO  date-1.
MOVE FUNCTION CONVERT-DATE-TIME
     ('95/05/30' DATE '%y/%m/%d' my-locale)
     TO  date-1.
MOVE FUNCTION CONVERT-DATE-TIME
     ('95/05/30' DATE LOCALE my-locale)
     TO  date-1.
```

COS

The COS function returns a numeric value that approximates the cosine of the angle or arc specified by the argument in radians.

The function type is numeric.

Format



argument-1

Must be class numeric.

The returned value is the approximation of the cosine of the argument, and is greater than or equal to -1 and less than or equal to +1.

CURRENT-DATE

The CURRENT-DATE function returns a 21-character alphanumeric value that represents the calendar date, time of day, and time differential from Greenwich Mean Time provided by the system on which the function is evaluated.

The function type is alphanumeric.

Format

Reading from left to right, the 21 character positions in the value returned can be interpreted as follows:

Character Positions
Contents

1-4
Four numeric digits of the year in the Gregorian calendar.

5-6
Two numeric digits of the month of the year, in the range 01 through 12.

7-8
Two numeric digits of the day of the month, in the range 01 through 31.

9-10
Two numeric digits of the hours past midnight, in the range 00 through 23.

11-12
Two numeric digits of the minutes past the hour, in the range 00 through 59.

13-14
Two numeric digits of the seconds past the minute, in the range 00 through 59.

15-16
Two numeric digits of the hundredths of a second past the second, in the range 00 through 99.

17
Either the character '-' or the character '+'. The character '-' is returned if the local time indicated in the previous character positions is behind Greenwich Mean Time. The character '+' is returned if the local time indicated is the same as or ahead of Greenwich Mean Time.

18-19
If character position 17 is '-', two numeric digits are returned in the range 00 through 12 indicating the number of hours that the reported time is behind Greenwich Mean Time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the reported time is ahead of Greenwich Mean Time.

20-21
Two numeric digits are returned in the range 00 through 59 indicating the number of additional minutes that the reported time is ahead of or behind Greenwich Mean Time, depending on whether character position 17 is '+' or '-', respectively.

For more information, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

DATE-OF-INTEGER

The DATE-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to standard date form (YYYYMMDD).

The function type is integer.

The function result is an eight-digit integer.

Format

argument-1
A positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar. The valid range is 1 to 3,067,671, which corresponds to dates ranging from January 1, 1601 through December 31, 9999.

The returned value represents the International Standards Organization (ISO) standard date equivalent to the integer specified as argument-1.

The returned value is an integer of the form YYYYMMDD where YYYY represents a year in the Gregorian calendar; MM represents the month of that year; and DD represents the day of that month.

DAY-OF-INTEGER

The DAY-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to Julian date form (YYYYDDD).

The function type is integer.

The function result is a seven-digit integer.

Format

argument-1
A positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar. The valid range is 1 to 3,067,671, which corresponds to dates ranging from January 1, 1601 through December 31, 9999.

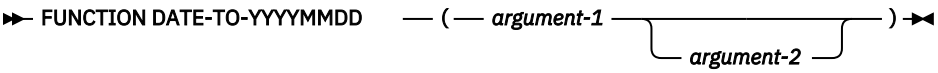
The returned value represents the Julian equivalent of the integer specified as argument-1. The returned value is an integer of the form YYYYDDD where YYYY represents a year in the Gregorian calendar and DDD represents the day of that year.

[IBM Extension] DATE-TO-YYYYMMDD

The DATE-TO-YYYYMMDD function converts argument-1 from the form YYnnnn to the form YYYYnnnn. Argument-2, when added to the year at the time of execution, defines the ending year of a 100-year interval, or sliding window, into which the year of argument-1 falls.

The type of the function is integer.

Format



argument-1
Must be a positive integer less than 1000000.

argument-2
Must be an integer. If argument-2 is omitted, the function will be evaluated with a value of 50 for argument-2. At the time of execution, the sum of the year with argument-2 will be less than 10000, and greater than 1699.

The equivalent arithmetic expression is:

```
(FUNCTION YEAR-TO-YYYY (YY, argument-2) * 10000 + nnnn)
```

where

```
YY = (argument-1/10000) truncated to an integer value
nnnn = argument-1 modulus 10000
```

This function supports a sliding window algorithm. See “YEAR-TO-YYYY” on page 504 for a discussion of how to specify a fixed window.

[End of IBM Extension]

Examples

In the year 2002, the returned value for:

```
FUNCTION DATE-TO-YYYYMMDD(851003, 10)
```

is 19851003.

In the year 1994, the returned value for:

```
FUNCTION DATE-TO-YYYYMMDD(981002,(-10))
```

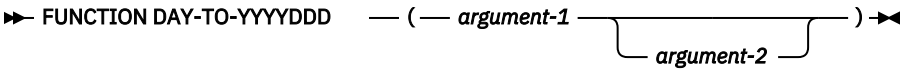
is 18981002.

[IBM Extension] DAY-TO-YYYYDDD

The DAY-TO-YYYYDDD function converts argument-1 from the form YYnnn to the form YYYYnnn. Argument-2, when added to the year at the time of execution, defines the ending year of a 100-year interval, or sliding window, into which the year of argument-1 falls.

The type of the function is integer.

Format



argument-1
Must be a positive integer less than 100000.

argument-2
Must be an integer. If argument-2 is omitted, the function will be evaluated with a value of 50 for argument-2. At the time of execution, the sum of the year with argument-2 will be less than 10000, and greater than 1699.

The equivalent arithmetic expression is:

```
(FUNCTION YEAR-TO-YYYY (YY, argument-2) * 1000 + nnn)
```

where

```
YY = (argument-1/1000) truncated to an integer value
nnn = argument-1 modulus 1000
```

This function supports a sliding window algorithm. See “YEAR-TO-YYYY” on page 504 for a discussion of how to specify a fixed window.

[End of IBM Extension]

Examples

In the year 1996, the returned value for:

```
FUNCTION DAY-TO-YYYYDDD(85003, 10)
```

is 1985003.

In the year 2013, the returned value for:

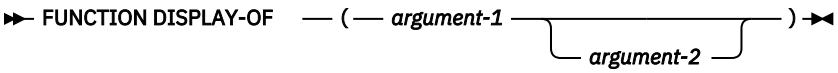
```
FUNCTION DAY-TO-YYYYDDD(95005,(-10))
```

is 1995005.

DISPLAY-OF

The DISPLAY-OF function returns an alphanumeric character string consisting of the content of argument-1 converted to a specific code page representation. The type of the function is alphanumeric.

Format 1: Specify target CCSID



argument-1
Must be of class national. Argument-1 identifies the source string for the conversion.

[IBM Extension] argument-2
Must be an integer. Argument-2 identifies the target code page for the conversion. Argument-2 must be a valid CCSID number identifying an EBCDIC, ASCII, UTF-8, or EUC code page. The EBCDIC or ASCII CCSID can identify a code page that is SBCS, DBCS, or mixed SBCS/DBCS. [End of IBM Extension]

If argument-2 is omitted, the target code page is the one in effect for the CCSID compiler option when the source code was compiled. If the target code page in effect is 65535, then default CCSID 37 will be used.

The returned value is an alphanumeric character string consisting of the characters of argument-1 converted to the target code page representation. When a source character cannot be converted to a character in the target code page, the source character is replaced with the system-defined substitution character X'3F' for a single-byte EBCDIC target code page and X'FEFE' for a DBCS code page. No exception condition is raised.

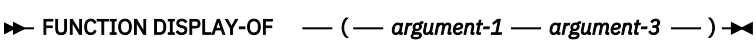
The length of the returned value depends on the content of argument-1 and the characteristics of the target code page.

Exceptions: If the conversion fails, a severe run-time error occurs. Verify that the conversion from the national data (coded using the UCS-2 CCSID specified in the National CCSID compiler option or in the NTLCCSID PROCESS option) to the target CCSID is supported on the operating system.

Usage notes

1. If the target code page is a mixed SBCS/DBCS EBCDIC code page, the returned value can include DBCS substrings delimited by shift-out and shift-in control characters.
2. The DISPLAY-OF function, with an integer argument-2 specified, can be used to generate character data represented in a code page that differs from that specified in the CCSID compiler option. Special care needs to be taken because subsequent COBOL operations on that data can involve implicit conversions that assume the data is represented in the EBCDIC code page specified in the CCSID compiler option.

Format 2: Specify user substitution character



argument-1
Must be of class national. Argument-1 identifies the source string for the conversion.

argument-3
Must be a literal or data item of class alphabetic, alphanumeric or DBCS, with one character position in length. Argument-3 specifies an alphanumeric or DBCS substitution character used in conversion of national characters for which there is no corresponding alphanumeric or DBCS character. User substitution character is only supported for EBCDIC and ASCII code pages. If it is used for another code page such as EUC, a compiler error message (of severity 30) will be issued and argument-3 will be ignored.

User substitution character has the same CCSID as the target code page CCSID in effect. If the target code page CCSID in effect is double byte, only double byte substitution character (DBCS) can be specified as argument-3. If the target code page CCSID in effect is single byte or mixed byte, only single byte substitution character (alphabetic or alphanumeric) can be specified as argument-3,

double byte substitution character for mixed byte CCSID can not be replaced by user substitution character. If inconsistency between the target CCSID type and argument-3 class is detected at compile time, a compiler error message (of severity 30) will be issued; if inconsistency is detected at runtime, a severe run-time message will be issued. If user answers the message with 'G' to continue the program, argument-3 will be ignored.

The target code page is the one in effect for the CCSID compiler option when the source code was compiled. If the CCSID compiler option in effect is 65535, then default CCSID 37 will be used.

The returned value is an alphanumeric character string consisting of the characters of argument-1 converted to the target code page representation. When a source character cannot be converted to a character in the target code page, the source character is replaced with the user substitution character argument-3. No exception condition is raised.

The length of the returned value depends on the content of argument-1 and the characteristics of the target code page.

Exceptions: If the conversion fails, a severe run-time error occurs. Verify that the conversion from the source ccsid (UCS-2) to the target CCSID is supported on the operating system.

You can nest the DISPLAY-OF and NATIONAL-OF intrinsic functions to easily convert from any CCSID to any other CCSID.

For example, the following code converts an EBCDIC string to an ASCII string:

```
  77  EBCDIC-CCSID PIC 9(4) BINARY VALUE 1140.
  77  ASCII-CCSID PIC 9(4) BINARY VALUE 819.
  77  Input-EBCDIC PIC X(80).
  77  ASCII-Output PIC X(80).
  . . .
* Convert EBCDIC to ASCII
      Move Function
        Display-of
          (Function National-of
             (Input-EBCDIC EBCDIC-CCSID)
            ASCII-CCSID
          )
      to ASCII-output
```

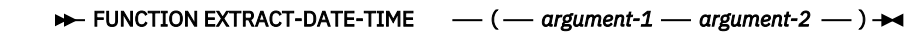
[IBM Extension] EXTRACT-DATE-TIME

The EXTRACT-DATE-TIME function returns part of a date, time, or timestamp item.

The function type is integer or alphanumeric. If argument-2 is a keyword (such as MONTHS or DAYS), or consists of only numeric specifiers, an integer is returned. Otherwise, an alphanumeric data item is returned.

The length of the result depends on the values extracted from the date-time item.

Format



argument-1
Must be a date, time, or timestamp item.

argument-2
Specifies the values to be returned by the EXTRACT-DATE-TIME function.

Argument-2 is a keyword that represents a duration or a non-numeric literal that contains one or more separators and conversion specifications.

If the non-numeric literal contains only numeric conversion specifiers, the value returned by the EXTRACT-DATE-TIME function is an integer. A non-numeric literal containing separators or alphanumeric conversion specifiers will result in an alphanumeric return value.

If argument-2 is a keyword, an integer is returned.

The valid durations and their equivalent conversion specifications are:

- YEARS ('@Y')
- MONTHS ('%m')
- DAYS ('%d')
- HOURS ('%H')
- MINUTES ('%M')
- SECONDS ('%S')
- MICROSECONDS ('@Sm').
- PICOSECONDS ('@Sp').

For a list of other valid conversion specifications see Table 5 on page 90 in the description of the FORMAT clause of the SPECIAL-NAMES paragraph.

The duration keyword or conversion specifier used must be consistent with argument-1. For example, the duration keywords must obey the following rules:

1. YEARS, MONTHS, and DAYS can only be extracted from a date or timestamp item.
2. HOURS, MINUTES, SECONDS, and MICROSECONDS can only be extracted from a time or timestamp item.
3. If argument-1 is a locale-based data item, and argument-2 contains locale-based conversion specifiers (such as %p), the locale-based conversion specifier (%p, in this case) uses the locale of argument-1.

   If argument-1 is *not* a locale-based data item, then the locale-based conversion specifier (%p, in this case) is treated as a non-locale-based conversion specifier and the % is replaced with an @. Using our example, this means that %p would become @p, where @p is the non-locale-based equivalent of %p.
4. PICOSECONDS can only be extracted from a timestamp item.

[End of IBM Extension]

Examples

The following examples show how the EXTRACT-DATE-TIME intrinsic function can be used:

```
COMPUTE integer-1 = FUNCTION EXTRACT-DATE-TIME (date-3 MONTHS).
COMPUTE integer-1 = FUNCTION EXTRACT-DATE-TIME (date-3 '%m').
MOVE FUNCTION EXTRACT-DATE-TIME (date-2 '%m/%d') to alphanum-1.
```

FACTORIAL

The FACTORIAL function returns an integer that is the factorial of the argument specified.

The function type is integer.

Format

argument-1

argument-1 If the *NOEXTEND compiler option is in effect, then argument-1 must be an integer greater than or equal to zero and less than or equal to 28. If the *EXTEND31 compiler option is in effect, then argument-1 must be an integer greater than or equal to zero and less than or equal to 29. If the value of argument-1 is zero, the value 1 is returned; otherwise, its factorial is returned. If the *EXTEND31FULL or *EXTEND63 compiler option is in effect, then argument-1 must be an integer greater than or equal to zero and less than or equal to 49.

[IBM Extension] FIND-DURATION

The FIND-DURATION function is used to calculate a duration between:

- Two dates
- A date and a timestamp
- Two times
- A time and a timestamp
- Two timestamps.

The FIND-DURATION function returns an integer in the form of complete units of the specified duration. Any rounding is done downwards. The calculation of durations includes microseconds.

The function type is integer.

The function result is a nine-digit integer. If the function result is larger than 9 digits (999,999,999), a machine check occurs.

Format

argument-1, argument-2
Must be a date, time, or timestamp item.

Argument-1 is subtracted from argument-2. The value returned is the number of complete units of argument-3. If argument-1 is later than argument-2, the result is negative. If argument-1 is earlier than argument-2, the result is positive.

argument-3
Is a keyword that represents a duration. The valid duration keywords are:

- YEARS
- MONTHS
- DAYS
- HOURS
- MINUTES
- SECONDS
- MICROSECONDS
- PICOSECONDS

In order to determine the valid duration keywords, the following rules apply:

1. If argument-1 or argument-2 is a date item, the duration specified must be consistent with a date.
2. If argument-1 or argument-2 is a time item, the duration specified must be consistent with a time.
3. If the returned value is not an integer, it is truncated. For example, the duration between March 17, 1997 and May 2, 1997 is 1.5 months. Since FIND-DURATION only returns an integer the .5 would be truncated, and the actual value returned would be 1.
4. PICOSECONDS duration can only be requested when argument-1 and argument-2 are timestamp items.

[End of IBM Extension]

Examples

The following examples show how the FIND-DURATION intrinsic function can be used:

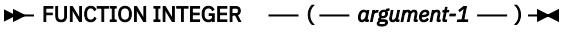
```
    COMPUTE integer-1 = FUNCTION FIND-DURATION (date-3 date-4 MONTHS).
    COMPUTE integer-1 = FUNCTION FIND-DURATION (timestamp-1 date-5 MONTHS).
```

INTEGER

The INTEGER function returns the greatest integer value that is less than or equal to the argument.

The function type is integer.

Format



argument-1
Must be class numeric.

The returned value is the greatest integer less than or equal to the value of argument-1. For example,

```
FUNCTION INTEGER (2.5)
```

will return a value of 2; and

```
FUNCTION INTEGER (-2.5)
```

will return a value of -3.

INTEGER-OF-DATE

The INTEGER-OF-DATE function converts a date in the Gregorian calendar from standard date form (YYYYMMDD) to integer date form.

The function type is integer.

The function result is a seven-digit integer with a range from 1 to 3,067,671.

Format

argument-1
Must be an integer of the form YYYYMMDD, whose value is obtained from the calculation (YYYY * 10,000) + (MM * 100) + DD.

- YYYY represents the year in the Gregorian calendar. It must be an integer greater than 1600, but not greater than 9999.
- MM represents a month and must be a positive integer less than 13.
- DD represents a day and must be a positive integer less than 32, provided that it is valid for the specified month and year combination.

The returned value is an integer that is the number of days the date represented by argument-1 succeeds December 31, 1600 in the Gregorian calendar.

INTEGER-OF-DAY

The INTEGER-OF-DAY function converts a date in the Gregorian calendar from Julian date form (YYYYDDD) to integer date form.

The function type is integer.

The function result is a seven-digit integer.

Format

argument-1
Must be an integer of the form YYYYDDD whose value is obtained from the calculation (YYYY * 1000) + DDD.

- YYYY represents the year in the Gregorian calendar. It must be an integer greater than 1600, but not greater than 9999.

- DDD represents the day of the year. It must be a positive integer less than 367, provided that it is valid for the year specified.

The returned value is an integer that is the number of days the date represented by argument-1 succeeds December 31, 1600 in the Gregorian calendar.

INTEGER-PART

The INTEGER-PART function returns an integer that is the integer portion of the argument specified.

The function type is integer.

Format

argument-1
Must be class numeric.

If the value of argument-1 is zero, the returned value is zero. If the value of argument-1 is positive, the returned value is the greatest integer less than or equal to the value of argument-1. If the value of argument-1 is negative, the returned value is the least integer greater than or equal to the value of argument-1.

For example,

```
FUNCTION INTEGER-PART (+1.5)
```

will return a value of +1; and

```
FUNCTION INTEGER-PART (-1.5)
```

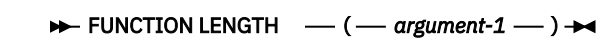
will return a value of -1.

LENGTH

The LENGTH function returns an integer equal to the length of the argument in bytes. The function type is integer.

The function result is a nine-digit integer.

Format



argument-1
Can be a nonnumeric, boolean, or DBCS literal; a data item of any class or category.

If argument-1, or any data item subordinate to argument-1, is described with the DEPENDING phrase of the OCCURS clause, the contents of the data item referenced by the data-name specified in the DEPENDING phrase are used at the time the LENGTH function is evaluated.

Argument-1 can be a type-name, or an item that is subordinate to a type-name.

A data item described with USAGE IS POINTER or USAGE IS PROCEDURE-POINTER can be used as argument-1 to the LENGTH function. The result will always be 16.

The ADDRESS OF special register, or the LENGTH OF special register, can be used as argument-1 to the LENGTH function. The result will always be 16 or 4 respectively, independent of the ADDRESS OF or LENGTH OF object.

If argument-1 is a nonnumeric literal, an elementary data item, or a group data item that does not contain a variable occurrence data item, the value returned is an integer equal to the length of argument-1 in character positions.

If argument-1 is a null-terminated nonnumeric literal, the returned value is equal to the number of alphanumeric character positions in the literal excluding the null character at the end of the literal. The length of a null-terminated nonnumeric literal containing a mix of single-byte and double-byte characters is counted as though each byte were a single-byte character.

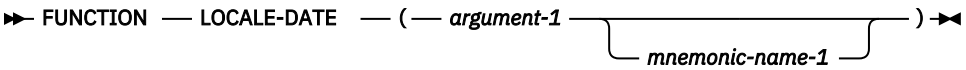
[IBM Extension] If argument-1 is a DBCS or national data item or literal, the returned value is the length of the argument in DBCS or national character positions. For example, `FUNCTION LENGTH (G"D1D2")` will return the value 2, not 4. [End of IBM Extension]

The returned value includes implicit FILLER characters, if any.

[IBM Extension] LOCALE-DATE

The LOCALE-DATE function returns a character string containing a date in a culturally-appropriate format specified by a locale.

The function type is alphanumeric.



argument-1
Argument-1 must be alphanumeric or DBCS, and 8 character positions in length.

Argument-1 must be a date in the same format as the year, month, and day returned in character positions 1 through 8 by the CURRENT-DATE function. For more information about the CURRENT-DATE function, refer to "CURRENT-DATE" on page 475.

mnemonic-name-1
Mnemonic-name-1 must be associated with a locale in the SPECIAL-NAMES paragraph.

[End of IBM Extension]

Returned Values

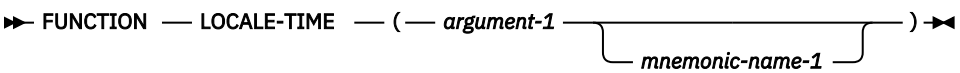
The following values are returned by the LOCALE-DATE intrinsic function:

1. If mnemonic-name-1 is specified, the locale used for formatting the date is the one associated with mnemonic-name-1; otherwise, the current locale is used. If the locale associated with mnemonic-name-1 is not available, an operating system escape message is issued.
2. The returned value is a character string containing the date specified by argument-1, and is returned in the date format specified in the locale.
3. The length of the returned value depends on the format specified in the locale.

[IBM Extension] LOCALE-TIME

The LOCALE-TIME function returns a character string containing a time in a culturally-appropriate format specified by a locale.

The function type is alphanumeric.



argument-1
Argument-1 must be alphanumeric or DBCS and must be 13 character positions in length.

The content of argument-1 must be in the same format as the hours, minutes, and seconds returned in character positions 9 through 21 by the CURRENT-DATE function. For more information about the CURRENT-DATE function, refer to "CURRENT-DATE" on page 475.

mnemonic-name-1
Mnemonic-name-1 must be associated with a locale in the SPECIAL-NAMES paragraph.

[End of IBM Extension]

Returned Values

The following values are returned by the LOCALE-TIME intrinsic function:

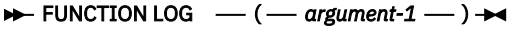
1. If mnemonic-1 is specified, the locale used for formatting the time is the one associated with mnemonic-name-1; otherwise, the current locale is used. If the locale associated with mnemonic-name-1 is not available, an operating system escape message is issued.
2. The returned value is a character-string containing the hours, minutes, and seconds of the time specified by argument-1 in the culturally-appropriate format indicated in the locale. The value will be adjusted for any difference between the offset from Universal Coordinated time (Greenwich Mean time) held in the last five character positions of argument-1, and that specified for the LC-TOD category of the locale.
3. The length of the returned value depends on the format indicated in the locale.

LOG

The LOG function returns a numeric value that approximates the logarithm to the base ***e*** (natural log) of the argument specified.

The function type is numeric.

Format



argument-1
Must be class numeric. The value of argument-1 must be greater than zero.

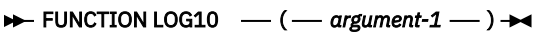
The returned value is the approximation of the logarithm to the base ***e*** of argument-1.

LOG10

The LOG10 function returns a numeric value that approximates the logarithm to the base 10 of the argument specified.

The function type is numeric.

Format



argument-1
Must be class numeric. The value of argument-1 must be greater than zero.

The returned value is the approximation of the logarithm to the base 10 of argument-1.

LOWER-CASE

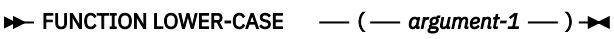
The LOWER-CASE function returns a character string that is the same length as the argument specified with each uppercase letter replaced by the corresponding lowercase letter.

The function type depends on the argument types, as follows:

Argument Type	Function Type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
DBCS[1]	DBCS[1]

Note: [1] IBM Extension

Format



argument-1
Must be class alphabetic, or alphanumeric and must be at least one character in length.

[IBM Extension] Argument-1 can be DBCS or national. [End of IBM Extension]

The same character string as argument-1 is returned, except that each uppercase letter is replaced by the corresponding lowercase letter. A program collating sequence or code page does not affect the returned value.

[IBM Extension] If argument-1 is DBCS, the DBCS value is not affected. If argument-1 is a mixed literal, only the single byte portions of the literal are affected. [End of IBM Extension]

The returned character string has the same length as argument-1.

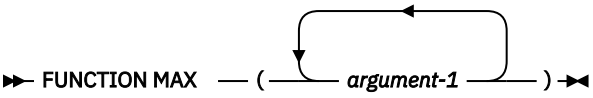
MAX

The MAX function returns the content of the argument that contains the maximum value.

The function type depends on the argument types, as follows:

Argument Type	Function Type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
National	National
Index	Index
All arguments integer	Integer
Numeric (some arguments can be integer)	Numeric

Format



argument-1
Can be class numeric, alphanumeric, alphabetic, or DBCS, and cannot be class boolean.

If more than one argument-1 is specified, the combination of alphabetic and alphanumeric arguments is allowed. Other combinations of argument types are not allowed. For example DBCS arguments can not be mixed with alphanumeric arguments.

The returned value is the content of argument-1 having the greatest value. The comparisons used to determine the greatest value are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 225.

If more than one argument has the same greatest value, the leftmost argument having that value is returned.

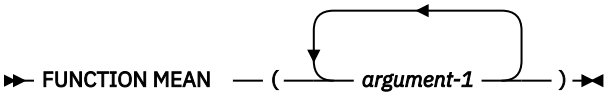
If the type of the function is alphanumeric or DBCS, the size of the returned value is the same as the size of the selected argument.

MEAN

The MEAN function returns a numeric value that approximates the arithmetic average of its arguments.

The function type is numeric.

Format



argument-1
Must be class numeric.

The returned value is the arithmetic mean of the argument-1 series. The returned value is defined as the sum of the argument-1 series divided by the number of occurrences referenced by argument-1.

The equivalent arithmetic expression shall be as follows:

1. For one occurrence of argument-1,

   (argument-1)
2. For two occurrences of argument-1,

   ((argument-1_1 + argument-1_2) / 2)
3. For n occurrences of argument-1,

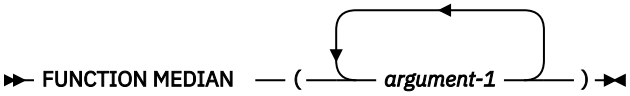
   ((argument-1_1 + argument-1_2 + ... + argument-1_n) / n)

MEDIAN

The MEDIAN function returns the content of the argument whose value is the middle value in the list formed by arranging the arguments in sorted order.

The function type is numeric.

Format



argument-1
Must be class numeric.

The returned value is the content of argument-1 having the middle value in the list formed by arranging all argument-1 values in sorted order.

If the number of occurrences referenced by argument-1 is odd, the returned value is such that at least half of the occurrences referenced by argument-1 are greater than or equal to the returned value and at least half are less than or equal. If the number of occurrences referenced by argument-1 is even, the returned value is the arithmetic mean of the values referenced by the two middle occurrences.

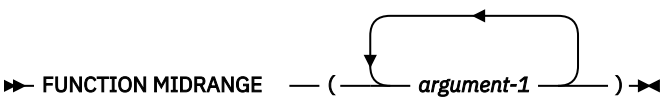
The comparisons used to arrange the argument values in sorted order are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 225.

MIDRANGE

The MIDRANGE function returns a numeric value that approximates the arithmetic average of the values of the minimum argument and the maximum argument.

The function type is numeric.

Format



argument-1
Must be class numeric.

The returned value is the arithmetic mean of the value of the greatest argument-1 and the value of the least argument-1. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions. For more information, see "Conditional Expressions" on page 225.

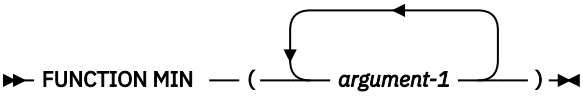
MIN

The MIN function returns the content of the argument that contains the minimum value.

The function type depends on the argument types, as follows:

Argument Type	Function Type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
National	National
Index	Index
All arguments integer	Integer
Numeric (some arguments can be integer)	Numeric

Format



argument-1
Can be class numeric, alphanumeric, alphabetic, or DBCS, and cannot be class boolean.

If more than one argument-1 is specified, the combination of alphabetic and alphanumeric arguments is allowed. Other combinations of argument types are not allowed. For example DBCS arguments can not be mixed with alphanumeric arguments.

The returned value is the content of argument-1 having the least value. The comparisons used to determine the least value are made according to the rules for simple conditions. For more information, see "Conditional Expressions" on page 225.

If more than one argument-1 has the same least value, the leftmost argument-1 having that value is returned.

If the type of the function is alphanumeric or DBCS, the size of the returned value is the same as the size of the selected argument-1.

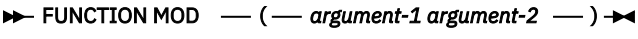
MOD

The MOD function returns an integer value that is argument-1 modulo argument-2.

The function type is integer.

The function result is an integer with as many digits as the shorter of argument-1 and argument-2.

Format



argument-1
Must be an integer.

argument-2
Must be an integer. Must not be zero.

The returned value is argument-1 modulo argument-2. The returned value is defined as:

argument-1 - (argument-2 * FUNCTION INTEGER (argument-1 / argument-2))

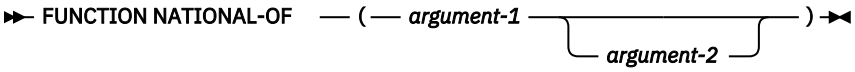
The following illustrates the expected results for some values of argument-1 and argument-2.

Argument-1	Argument-2	Return
11	5	1
-11	5	4
11	-5	-4
-11	-5	-1

NATIONAL-OF

The NATIONAL-OF function returns a national character string consisting of the UCS-2 representation of the characters in argument-1. The type of the function is national.

Format 1: Specify source CCSID



argument-1
Must be of class alphabetic, alphanumeric, or DBCS. Argument-1 identifies the source string for the conversion.

[IBM Extension] argument-2

Must be an integer. Argument-2 identifies the source code page for the conversion. Argument-2 must be a valid CCSID number identifying an EBCDIC, ASCII, UTF-8, or EUC code page. The EBCDIC or ASCII CCSID can identify a code page that is SBCS, DBCS, or mixed SBCS/DBCS.

If argument-2 is omitted, the source code page is the one in effect for the CCSID compiler option when the source code was compiled. If the source code page is 65535, then default CCSID 37 will be used.

[End of IBM Extension]

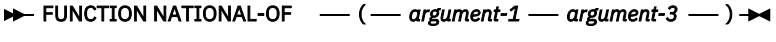
The returned value is a national character string consisting of the characters of argument-1 converted to national character representation (UCS-2 CCSID specified in the National CCSID compiler option or in the NTLCCSID PROCESS option). See "Conversions and Precision" in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information.

When a source character cannot be converted to a national character, the source character is converted to the system-defined substitution character X'FFFD'. No exception condition is raised.

The length of the returned value depends on the content of argument-1 and the characteristics of the source code page.

Exceptions: If the conversion fails, a severe run-time error occurs. Verify that the conversion from the source CCSID to the target CCSID (CCSID specified in the National CCSID compiler option or in the NTLCCSID PROCESS option) is supported on the operating system.

Format 2: Specify user substitution character



argument-1
Must be of class alphabetic, alphanumeric, or DBCS. Argument-1 identifies the source string for the conversion.

argument-3
Must be a national literal or national data item with one character position in length.

Argument-3 specifies a national substitution character used in conversion of alphanumeric characters for which there is no corresponding national character.

The source code page is the one in effect for the CCSID compiler option when the source code was compiled. If the CCSID compiler option is 65535, then default CCSID 37 will be used.

The returned value is a national character string consisting of the characters of argument-1 converted to national character representation (CCSID specified in the National CCSID compiler option or in the NTLCCSID PROCESS option). When a source character cannot be converted to a national character, the source character is converted to the user substitution character argument-3. No exception condition is raised.

The length of the returned value depends on the content of argument-1 and the characteristics of the source code page.

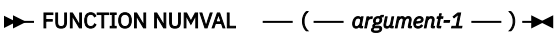
Exceptions: If the conversion fails, a severe run-time error occurs. Verify that the conversion from the source ccsid to the target CCSID (UCS-2) is supported on the operating system.

NUMVAL

The NUMVAL function returns the numeric value represented by the alphanumeric character string specified in an argument. The function strips away any leading or trailing blanks in the string, producing a numeric value that can be used in an arithmetic expression.

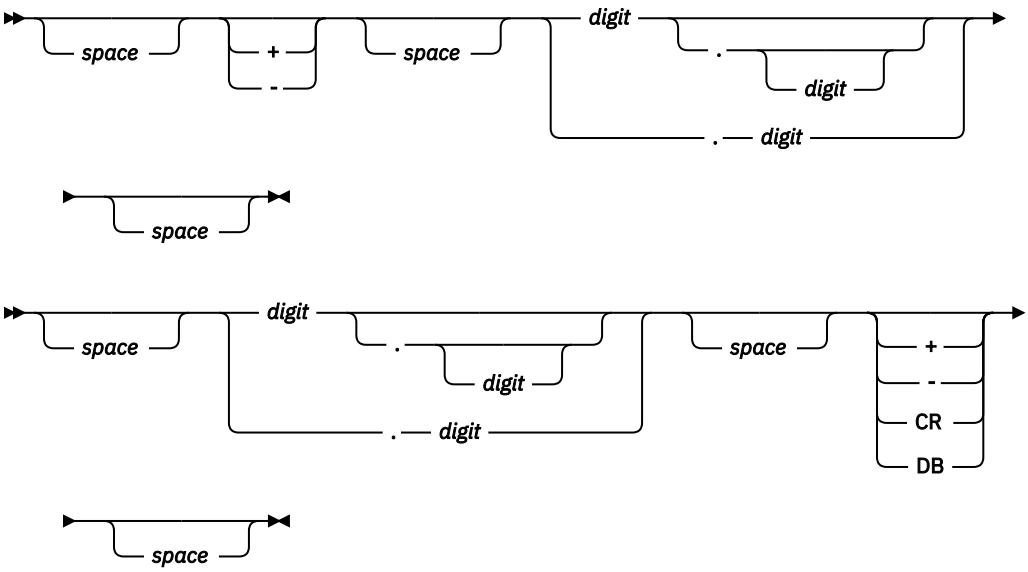
The function type is numeric.

Format



argument-1

Must be a nonnumeric literal or an alphanumeric data item whose content has the following formats:



space

A string of one or more spaces.

digit

A string of one or more digits. The total number of digits must not exceed 18.

If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, a comma must be used in argument-1 rather than a decimal point.

The returned value is a floating-point approximation of the numeric value represented by argument-1. See "Conversions and Precision" in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information.

NUMVAL-C

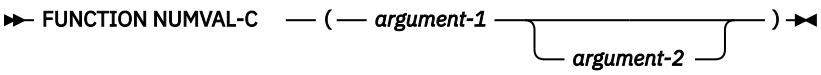
The NUMVAL-C function returns the numeric value represented by the alphanumeric character string specified as argument-1. Any optional currency sign specified by argument-2 and any optional commas preceding the decimal point are stripped away, producing a numeric value that can be used in an arithmetic expression.

The NUMVAL-C function may not be specified under the following conditions:

- More than one CURRENCY SIGN clause is specified within the program
- The WITH PICTURE SYMBOL phrase is specified in a CURRENCY SIGN clause
- A lowercase letter is specified as the currency symbol

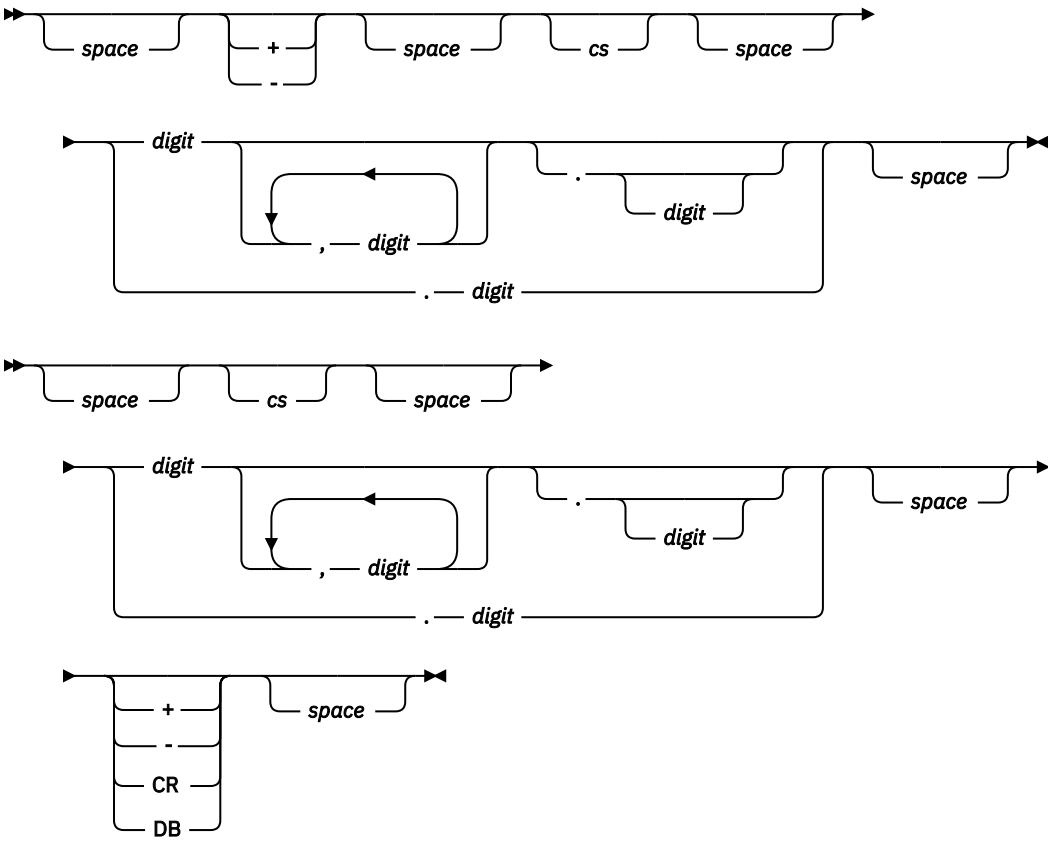
The function type is numeric.

Format



argument-1

Must be a nonnumeric literal or an alphanumeric data item whose content has the following formats:



space

A string of one or more spaces.

cs

The string of one or more characters specified by argument-2. At most, one copy of the characters specified by cs can occur in argument-1.

digit
A string of one or more digits. The total number of digits must not exceed 18.

If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, the functions of the comma and decimal point in argument-1 are reversed.

argument-2
If specified, must be a nonnumeric literal or alphanumeric data item, subject to the following rules:

- It must not contain any of the digits 0 through 9, any leading or trailing spaces, or any of the + - . , special characters.
- It can be of any length valid for an elementary or group data item, including zero.
- Matching of argument-2 is case-sensitive. For example, if you specify argument-2 as 'Dm', it will not match 'DM', 'dm' or 'dM'.

If argument-2 is not specified, the character used for cs is the currency symbol specified for the program.

The returned value is a floating-point approximation of the numeric value represented by argument-1. See "Conversions and Precision" in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information.

ORD

The ORD function returns an integer value that is the ordinal position of its argument in the collating sequence for the program. The lowest ordinal position is 1.

The function type is integer.

Format

argument-1
Must be one character in length and must be class alphabetic or alphanumeric.

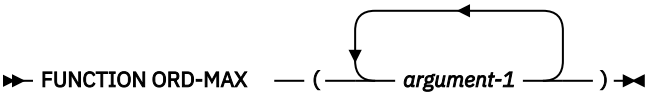
The returned value is the ordinal position of argument-1 in the collating sequence for the program; it ranges from 1 to 256 depending on the collating sequence.

ORD-MAX

The ORD-MAX function returns a value that is the ordinal number position, in the argument list, of the argument that contains the maximum value.

The function type is integer.

Format



argument-1
Must be class numeric, alphanumeric, or alphabetic (Class boolean is not allowed).

If more than one argument-1 is specified, all arguments must be of the same class with the exception that the combination of alphabetic and alphanumeric arguments is allowed.

The returned value is the ordinal number that corresponds to the position of argument-1 having the greatest value in the argument-1 series.

The comparisons used to determine the greatest valued argument-1 are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 225.

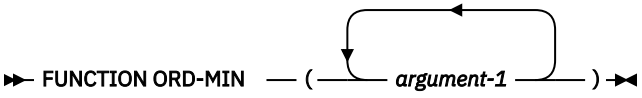
If more than one argument-1 has the same greatest value, the number returned corresponds to the position of the leftmost argument-1 having that value.

ORD-MIN

The ORD-MIN function returns a value that is the ordinal number of the argument that contains the minimum value.

The function type is integer.

Format



argument-1
Must be class numeric, alphanumeric, or alphabetic (Class boolean is not allowed).

If more than one argument-1 is specified, all arguments must be of the same class with the exception that the combination of alphabetic and alphanumeric arguments is allowed.

The returned value is the ordinal number that corresponds to the position of the argument-1 having the least value in the argument-1 series.

The comparisons used to determine the least valued argument-1 are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 225.

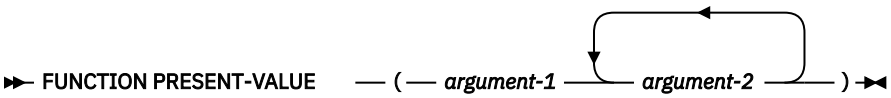
If more than one argument-1 has the same least value, the number returned corresponds to the position of the leftmost argument-1 having that value.

PRESENT-VALUE

The PRESENT-VALUE function returns a value that approximates the present value of a series of future period-end amounts specified by argument-2 at a discount rate specified by argument-1.

The function type is numeric.

Format



argument-1
Must be class numeric. Must be greater than -1.

argument-2
Must be class numeric.

The equivalent arithmetic expression for the returned value is:

1. For one occurrence of argument-2,

   (argument-2 / (1 + argument-1))

2. For two occurrences of argument-2,

   (argument-2_1 / (1 + argument-1) + argument-2_2 / (1 + argument-1)**2)

3. For n occurrences of argument-2,

   FUNCTION SUM (argument-2_1 / (1 + argument-1) ... argument-2_n / (1 + argument-1)**n)

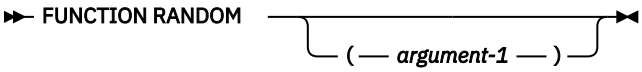
There is one term for each occurrence of argument-2. The exponent, n, is incremented from one by one for each term in the series.

RANDOM

The RANDOM function returns a numeric value that is a pseudo-random number from a rectangular distribution.

The function type is numeric.

Format



argument-1
If argument-1 is specified, it must be zero or a positive integer, up to and including (10**18)-1 which is the maximum value that can be specified in a PIC9(18) fixed item; however, only those in the range from zero up to and including 2,147,483,645 will yield a distinct sequence of pseudorandom numbers.

If a subsequent reference specifies argument-1, a new sequence of pseudo-random numbers is started.

If the first reference to this function in the run unit does not specify argument-1, the seed value used will be zero.

In each case, subsequent references without specifying argument-1 return the next number in the current sequence.

The returned value is exclusively between zero and one.

For a given seed value, the sequence of pseudorandom numbers will always be the same.

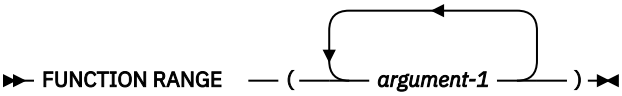
RANGE

The RANGE function returns a value that is equal to the value of the maximum argument minus the value of the minimum argument.

The function type depends on the argument types, as follows:

Argument Type	Function Type
All arguments integer	Integer
Numeric (some arguments can be integer)	Numeric

Format



argument-1
Must be class numeric.

The returned value is equal to argument-1 with the greatest value minus the argument-1 with the least value. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions. For more information, see “Conditional Expressions” on page 225.

The equivalent arithmetic expression for the RANGE function is:

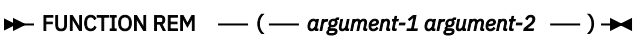
```
(FUNCTION MAX (argument-1) - FUNCTION MIN (argument-1))
```

REM

The REM function returns a numeric value that is the remainder of argument-1 divided by argument-2.

The function type is numeric.

Format



argument-1
Must be class numeric

argument-2
Must be class numeric. Must not be zero.

The returned value is the remainder of argument-1 divided by argument-2. It is defined as the expression:

```
argument-1 - (argument-2 * FUNCTION INTEGER-PART (argument-1/argument-2))
```

REVERSE

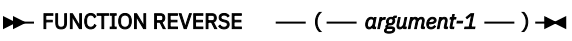
The REVERSE function returns a character string that is exactly the same length as the argument, whose characters are exactly the same as those specified in the argument, except that they are in reverse order.

The function type depends on the argument types, as follows:

Argument Type	Function Type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
DBCS[1]	DBCS[1]

Note: [1] IBM Extension

Format



argument-1
Must be class alphabetic or alphanumeric, and must be at least one character in length.

[IBM Extension] Argument-1 can be DBCS or national. [End of IBM Extension]

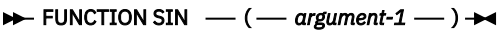
If argument-1 is a character string of length n, the returned value is a character string of length n such that, for 1 <= j <= n, the character in position j of the returned value is the character from position n-j+1 of argument-1.

SIN

The SIN function returns a numeric value that approximates the sine of the angle or arc specified by the argument in radians.

The function type is numeric.

Format



argument-1
Must be class numeric.

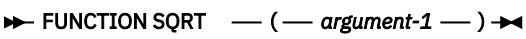
The returned value is the approximation of the sine of argument-1 and is greater than or equal to -1 and less than or equal to +1.

SQRT

The SQRT function returns a numeric value that approximates the square root of the argument specified.

The function type is numeric.

Format



argument-1
Must be class numeric. The value of argument-1 must be zero or positive.

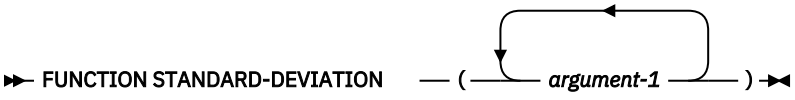
The returned value is the absolute value of the approximation of the square root of argument-1.

STANDARD-DEVIATION

The STANDARD-DEVIATION function returns a numeric value that approximates the standard deviation of its arguments.

The function type is numeric.

Format



argument-1
Must be class numeric.

The returned value is the approximation of the standard deviation of the argument-1 series. The returned value is calculated as follows:

1. The difference between each argument-1 and the arithmetic mean of the argument-1 series is calculated and squared.
2. The values obtained are then added together. This quantity is divided by the number of values in the argument-1 series.
3. The square root of the quotient obtained is then calculated. The returned value is the absolute value of this square root.

If the argument-1 series consists of only one value, or if the argument-1 series consists of all variable occurrence data items and the total number of occurrences for all of them is one, the returned value is zero.

The equivalent arithmetic expression for FUNCTION STANDARD-DEVIATION is:

```
FUNCTION SQRT (FUNCTION VARIANCE (argument-1))
```

[IBM Extension] SUBTRACT-DURATION

The SUBTRACT-DURATION function subtracts a duration from a date, time, or timestamp item, and returns the modified item.

The function type is date-time.

The length of the return value depends on the length of the date, time, or timestamp item specified in argument-1. The returned value will be truncated to the length of argument-1.

If a duration is subtracted from a date item, the date returned must fall within a certain range:

- For 4-digit dates, the range must be 0001/01/01 through 9999/12/31
- For 2-digit dates, the range must be 0001/01/01 through 9999/12/31, but the year is truncated to 2 digits
- For a 3-digit year (a 1-digit century and a 2-digit year), the range must be 1900/01/01 through 2899/12/31 (the default). This range can be changed by specifying the DATTIM PROCESS statement option.

If a duration is subtracted from a 2-digit date item, the range is the same as for a 4-digit year, but the year in the value returned is truncated to 2 digits.

Format

argument-1
Must be a date, time, or timestamp item.

Argument-1 is the value from which a duration is subtracted. The duration is specified in argument-2 and argument-3.

argument-2
Argument-2 is a keyword that represents a duration. The valid durations are:

- YEARS
- MONTHS
- DAYS
- HOURS
- MINUTES
- SECONDS
- MICROSECONDS
- PICOSECONDS

The duration keyword or conversion specifier used must be consistent with argument-1. For example, the duration keywords must obey the following rules:

1. YEARS, MONTHS, and DAYS can only be subtracted from a date or timestamp item.
2. HOURS, MINUTES, SECONDS, and MICROSECONDS can only be subtracted from a time or timestamp item.
3. PICOSECONDS can only be subtracted from a timestamp item.

argument-3
Must be an integer arithmetic expression. Argument-3 is the number of units of the duration, as specified in argument-2, that are to be subtracted from argument-1.

Argument-2 and argument-3 can be repeated. There should be no duplicate argument-2 in one intrinsic function.

Argument-3 can be a negative integer, but the function only takes its absolute value. If argument-3 is longer than 9 digits, it is truncated.

If a duration is subtracted from a date, and the result is invalid, the date is adjusted. For example, if a duration of 1 month is subtracted from the date `May 31, 1997`, the result would be the invalid date `April 31, 1997`. This date would be adjusted to the valid date `April 30, 1997`.

[End of IBM Extension]

Examples

The following examples show how the SUBTRACT-DURATION intrinsic function can be used:

```
MOVE FUNCTION SUBTRACT-DURATION (date-1 MONTHS 1)
     TO date-2.
MOVE FUNCTION SUBTRACT-DURATION (date-2 MONTHS 1 + 2 * 3)
     TO date-1.
MOVE FUNCTION SUBTRACT-DURATION (date-3 MONTHS 5 DAYS 1000)
     TO date-1.
```

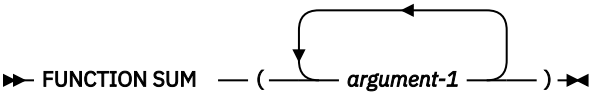
SUM

The SUM function returns a value that is the sum of the arguments.

The function type depends on the argument types, as follows:

Argument Type	Function Type
All arguments integer	Integer
Numeric (some arguments can be integer)	Numeric

Format



argument-1
Must be class numeric.

The returned value is the sum of the arguments. If the argument-1 series are all integers, the value returned is an integer. If the argument-1 series are not all integers, a numeric value is returned.

The equivalent arithmetic expression is:

1. For one occurrence of argument-1,

   (argument-1)
2. For two occurrences of argument-1,

   (argument-1_1 + argument-1_2)
3. For n occurrences of argument-1,

   (argument-1_1 + argument-1_2 + ... + argument-1_n)

TAN

The TAN function returns a numeric value that approximates the tangent of the angle or arc that is specified by the argument in radians.

The function type is numeric.

Format

argument-1
Must be class numeric.

The returned value is the approximation of the tangent of argument-1.

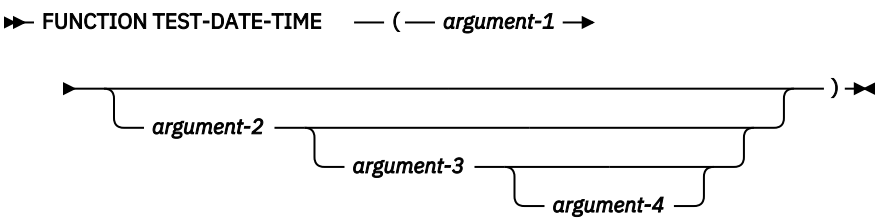
[IBM Extension] TEST-DATE-TIME

The TEST-DATE-TIME function takes a date, time, or timestamp item; alphanumeric item; numeric packed or zoned item; and determines if it is a valid date, time or timestamp. It returns true (B'1') if it is a valid item or false (B'0') if it is not a valid item.

The function type is boolean.

The length of the returned value is 1 byte.

Format



argument-1
Can be:

- A date, time, or timestamp item
- An item of class alphanumeric
- A non-numeric literal
- An item of class numeric integer.

If argument-1 *is* a date, time, or timestamp item, then argument-2 through argument-4 are optional. If argument-1 *is not* a date, time, or timestamp item, argument-2 must be specified (argument-3 and argument-4 are optional).

Argument-1 is tested to see if it is a valid item, based on its type or on argument-2 through argument-4.

argument-2
Must be one of the following keywords:

- DATE
- TIME
- TIMESTAMP.

If argument-2 is TIMESTAMP, argument-3 can only be specified with FORMAT OF special register, and argument-4 cannot be specified.

argument-3
Specifies the format of a date or time item. It must be:

- A nonnumeric literal at least 2 characters long
- The keyword LOCALE
- The FORMAT OF special register.

For a list of valid literals see the SPECIAL-NAMES FORMAT clause.

If argument-3 is the keyword LOCALE, then the format of the date or time is based on a LOCALE. If argument-4 is not specified, the current locale is used, otherwise the locale associated with the mnemonic-name or the LOCALE OF special register is used.

If argument-3 is not specified, the format used for the test is the one defined in the SPECIAL-NAMES FORMAT clause. For TIMESTAMP, if argument-3 is not specified, the default format of @Y-%m-%d-%H%M%S.@Sm is used.

argument-4
Must be a mnemonic-name associated with a LOCALE, or the LOCALE OF special register.

Argument-4 must follow these rules:

- If argument-4 is specified and argument-3 is a locale-based format literal, for example contains %p, then the locale-based format literal would use the locale specified in argument-4 to determine the actual value of the conversion specifiers.

- If argument-3 is a locale-based format literal (for example, contains %p) and argument-4 is not specified, the locale-based format literal would use the current locale to determine the actual value of the conversion specifiers.
- If argument-3 is a locale-based format literal (for example, contains %p), and the LOCALE OF special register is used to refer to a non-locale item, the locale-based format literal would use the default locale to determine the actual value of the conversion specifiers.

[End of IBM Extension]

Examples

The following examples show how the TEST-DATE-TIME intrinsic function can be used:

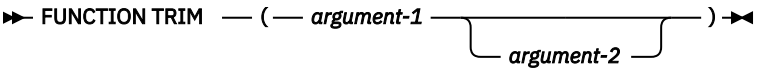
```
WORKING-STORAGE SECTION.
01 mydate1 PIC X(8) VALUE '07312013'.

PROCEDURE DIVISION.

IF FUNCTION TEST-DATE-TIME (mydate1 DATE '%d%m%Y') = B'1'
   DISPLAY 'Valid Date'
END-IF.
```

[IBM Extension] TRIM

The TRIM function returns the given string with any leading and trailing blanks removed, or the given string with any leading and trailing specified characters removed.

The type of the function is alphanumeric, DBCS or national depending on the class of its argument.

Format:



argument-1
Must be a nonnumeric literal, or data item of class alphabetic, alphanumeric, DBCS or national. Argument-1 identifies the source string for the trim.

argument-2
If specified, it must be a nonnumeric literal, or data item of the same class as argument-1. It specifies the characters to trim off. If not specified, the trim character defaults to blank.

If argument-2 is not specified, the returned value is an alphanumeric, DBCS or national character string consisting of the characters of argument-1 with any leading and trailing blanks removed. The blank character is a one byte space character (' ' or X'40') when argument-1 is of class alphanumeric, or one double-byte space (X'4040') when argument-1 is of class DBCS, or one national space (X'0020' or X'3000') when argument-1 is of class national.

If argument-2 is specified, all characters in argument-2 will be trimmed off from both ends of the string. The returned value is an alphanumeric, DBCS or national character string consisting of the characters of argument-1 with any leading and trailing characters specified in argument-2 removed.

The length of the returned string depends on the content and the class of argument-1. It is the length of the returned string in number of character positions. If argument-1 is a DBCS or national data item, then the length is in DBCS or national character positions.

[End of IBM Extension]

Returned Values

The order of the characters in the argument-2 parameter does not affect the outcome of the operation. The characters are a list of single characters. For example, `FUNCTION TRIML(fld, "abc")` will return the substring of fld that begins with any character that is not 'a', 'b', or 'c'. If fld contains "caxyz", `FUNCTION TRIM(fld, "abc")` will return "xyz". Characters can appear twice in the second parameter

with no error. For example, `FUNCTION TRIM(fld, "aba")` is valid. This means the same as `FUNCTION TRIM(fld, "ab")`.

If the second parameter of FUNCTION TRIM, TRIML or TRIMR is specified, blanks are not trimmed unless a blank appears as part of argument-2. TRIM, TRIML and TRIMR functions are not sensitive to mixed SBCS/DBCS strings, both argument-1 and argument-2 will be treated as SBCS if their class is alphanumeric.

Examples:

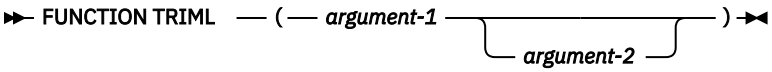
```
FUNCTION TRIM("xxxABxCxxx", "x") // returns 'ABxC'
FUNCTION TRIMR(">>>>ABC<<<<<", "<>") // returns '>>>>ABC'
MOVE "xyz" TO tc.
FUNCTION TRIML("xxyyzzyyzzABCxyzyxzxy", tc) // returns 'ABCxyzyxzxy'
```

[IBM Extension] TRIML

The TRIML function returns the given string with any leading blanks removed, or the given string with any leading specified characters removed.

The type of the function is alphanumeric, DBCS or national depending on the class of its argument.

Format:



argument-1
Must be a nonnumeric literal, or data item of class alphabetic, alphanumeric, DBCS or national. Argument-1 identifies the source string for the trim.

argument-2
If specified, it must be a nonnumeric literal, or data item of the same class as argument-1. It specifies the characters to trim off. If not specified, the trim character defaults to blank.

If argument-2 is not specified, the returned value is an alphanumeric, DBCS or national character string consisting of the characters of argument-1 with any leading blanks removed. The blank character is a one byte space character (' ' or X'40') when argument-1 is of class alphanumeric, or one double-byte space (X'4040') when argument-1 is of class DBCS, or one national space (X'0020' or X'3000') when argument-1 is of class national.

If argument-2 is specified, the returned value is an alphanumeric, DBCS or national character string consisting of the characters of argument-1 with any leading characters specified in argument-2 removed.

The length of the returned string depends on the content and the class of argument-1. It is the length of the returned string in number of character positions. If argument-1 is a DBCS or national data item, then the length is in DBCS or national character positions.

For more information on returned values and examples, see "TRIM" on page 500.

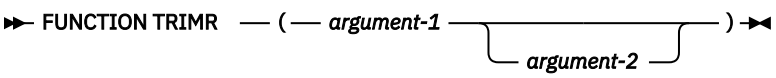
[End of IBM Extension]

[IBM Extension] TRIMR

The TRIMR function returns the given string with any trailing blanks removed, or the given string with any trailing specified characters removed.

The type of the function is alphanumeric, DBCS or national depending on the class of its argument.

Format:



argument-1
Must be a nonnumeric literal, or data item of class alphabetic, alphanumeric, DBCS or national. Argument-1 identifies the source string for the trim.

argument-2
If specified, it must be a nonnumeric literal, or data item of the same class as argument-1. It specifies the characters to trim off. If not specified, the trim character defaults to blank.

If argument-2 is not specified, the returned value is an alphanumeric, DBCS or national character string consisting of the characters of argument-1 with any trailing blanks removed. The blank character is a one byte space character (' ' or X'40') when argument-1 is of class alphanumeric, or one double-byte space (X'4040') when argument-1 is of class DBCS, or one national space (X'0020' or X'3000') when argument-1 is of class national.

If argument-2 is specified, the returned value is an alphanumeric, DBCS or national character string consisting of the characters of argument-1 with any trailing characters specified in argument-2 removed.

The length of the returned string depends on the content and the class of argument-1. It is the length of the returned string in number of character positions. If argument-1 is a DBCS or national data item, then the length is in DBCS or national character positions.

For more information on returned values and examples, see "TRIM" on page 500.

[End of IBM Extension]

UPPER-CASE

The UPPER-CASE function returns a character string that is the same length as the argument specified, with each lowercase letter replaced by the corresponding uppercase letter.

The function type depends on the argument types, as follows:

Argument Type	Function Type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
DBCS[1]	DBCS[1]

Note: [1] IBM Extension

Format

argument-1
Must be class alphabetic or alphanumeric, and must be at least one character in length.

[IBM Extension] Argument-1 can be DBCS or national. [End of IBM Extension]

The same character string as argument-1 is returned, except that each lowercase letter is replaced by the corresponding uppercase letter. A program collating sequence or code page does not affect the returned value.

[IBM Extension] If argument-1 is DBCS, the DBCS value is not affected. If argument-1 is a mixed literal, only the single byte portions of the literal are affected. [End of IBM Extension]

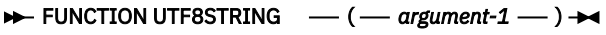
The character string returned has the same length as argument-1.

[IBM Extension] UTF8STRING

The UTF8STRING function converts the argument specified into the corresponding UTF-8 string. The string returned has variable length. Users are advised to allow sufficient length for the receiving argument returned by this function. The maximum length returns is twice the length of the original argument.

The function type is alphanumeric.

Format



argument-1
Must be alphabetic, alphanumeric, DBCS or national, and must be at least one character in length.

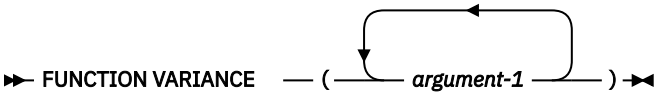
[End of IBM Extension]

VARIANCE

The VARIANCE function returns a numeric value that approximates the variance of its arguments.

The function type is numeric.

Format



argument-1
Must be class numeric.

The returned value is the approximation of the variance of the argument-1 series.

The returned value is defined as the square of the standard deviation of the argument-1 series. This value is calculated as follows:

1. The difference between each argument-1 value and the arithmetic mean of the argument-1 series is calculated and squared.
2. The values obtained are then added together. This quantity is divided by the number of values in the argument series.

If the argument-1 series consists of only one value, or if the argument-1 series consists of all variable occurrence data items and the total number of occurrences for all of them is one, the returned value is zero.

The equivalent arithmetic expression for FUNCTION VARIANCE is:

1. For one occurrence of argument-1,

   (0)

2. For two occurrences of argument-1,

   (((argument-1_1 - FUNCTION MEAN (argument-1)) ** 2 + (argument-1_2 - FUNCTION MEAN (argument-1)) ** 2) / 2)

3. For n occurrences of argument-1,

   (FUNCTION SUM (((argument-1_1 - FUNCTION MEAN (argument-1)) ** 2) ... ((argument-1_n - FUNCTION MEAN (argument-1)) ** 2)) / n)

WHEN-COMPILED

The WHEN-COMPILED function returns the date and time the program was compiled as provided by the system on which the program was compiled.

The function type is alphanumeric.

Format

Reading from left to right, the 21 character positions in the value returned can be interpreted as follows:

Character Positions	Contents
1-4	Four numeric digits of the year in the Gregorian calendar.
5-6	Two numeric digits of the month of the year, in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range 00 through 23.
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric digits of the seconds past the minute, in the range 00 through 59.
15-16	Two numeric digits of the hundredths of a second past the second, in the range 00 through 99.
17	Either the character '-' or the character '+'. The character '-' is returned if the local time indicated in the previous character positions is behind Greenwich Mean Time. The character '+' is returned if the local time indicated is the same as or ahead of Greenwich Mean Time.
18-19	If character position 17 is '-', two numeric digits are returned in the range 00 through 12 indicating the number of hours that the reported time is behind Greenwich Mean Time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the reported time is ahead of Greenwich Mean Time.
20-21	Two numeric digits are returned in the range 00 through 59 indicating the number of additional minutes that the reported time is ahead of or behind Greenwich Mean Time, depending on whether character position 17 is '+' or '-', respectively.

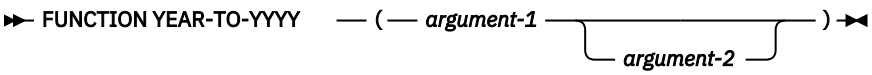
The returned value is the date and time of compilation of the source program that contains this function. For ILE COBOL, the date and time is calculated at the beginning of the compile and is placed in the header line of each listing page, on the DATE-COMPILED paragraph, and in the WHEN-COMPILED special-register. If the program is a contained program, the returned value is the compilation date and time associated with the separately compiled program in which it is contained.

[IBM Extension] YEAR-TO-YYYY

The YEAR-TO-YYYY function converts argument-1, the two low-order digits of a year, to a four-digit year. Argument-2, when added to the year at the time of execution, defines the ending year of a 100-year interval, or sliding window, into which the year of argument-1 falls.

The type of the function is integer.

Format



argument-1
Must be a nonnegative integer that is less that 100.

argument-2
Must be an integer. If argument-2 is omitted, the function will be evaluated with a value of 50 for argument-2. At the time of execution, the sum of the year and argument-2 will be less than 10000, and greater than 1699.

In order for the compiler to calculate the FUNCTION YEAR-TO-YYYY, the ending year of the sliding window (or "maximum-year") needs to be calculated first. The "maximum-year" is calculated as follows:

```
(FUNCTION NUMVAL(FUNCTION CURRENT-DATE(1:4)) + argument-2)
```

Given that the "maximum-year" above the FUNCTION YEAR-TO-YYYY is equivalent to one of the two arithmetic expressions depending on the following condition:

```
maximum-year modulus 100 >= argument-1
```

When this condition is true, the equivalent arithmetic expression is:

```
(argument-1 + 100 * (truncated integer value of (maximum-year/100)))
```

Otherwise, the equivalent arithmetic expression is:

```
(argument-1 + 100 * (truncated integer value of (maximum-year/100) - 1))
```

The YEAR-TO-YYYY function implements a sliding window algorithm. To use it for a fixed window, argument-2 can be specified as follows, where fixed-maximum-year is the maximum year in the fixed 100-year intervals.

If the fixed window is 1973 through 2072, then in 2009 argument-2 will have the value of 63, and in 2019, the value of 53.

```
(fixed-maximum-year - FUNCTION NUMVAL (FUNCTION CURRENT-DATE (1:4)))
```

[End of IBM Extension]

Examples

In the year 1995, the returned value for:

```
FUNCTION YEAR-TO-YYYY(4, 23)
```

is 2004.

In the year 2008, the returned value for:

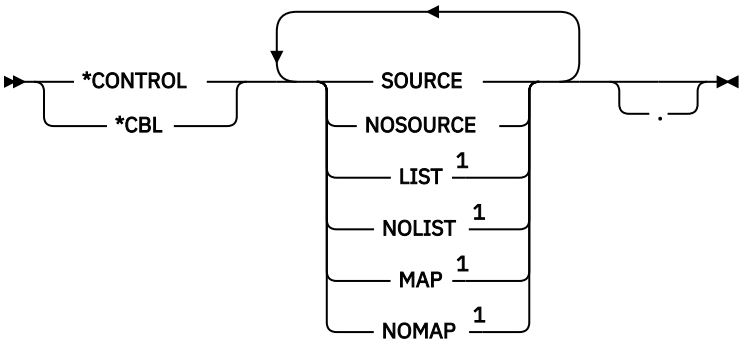
```
FUNCTION YEAR-TO-YYYY(98,(-15))
```

is 1898.

Chapter 8. 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.



***CONTROL (*CBL) Statement - Format**

Notes:

[1] Syntax-checked only.

For a complete discussion of compiler output, see the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

The *CONTROL and *CBL statements are synonymous.

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 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 *IBM Rational Development Studio for i: ILE COBOL Programmer'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 a sequence of source programs.

[End of IBM Extension]

*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.

COPY Statement

The COPY statement is a library statement that places prewritten text in a COBOL program.

[IBM Extension]

- Format 2 - DDS Translate
- Format 3 - Basic IFS

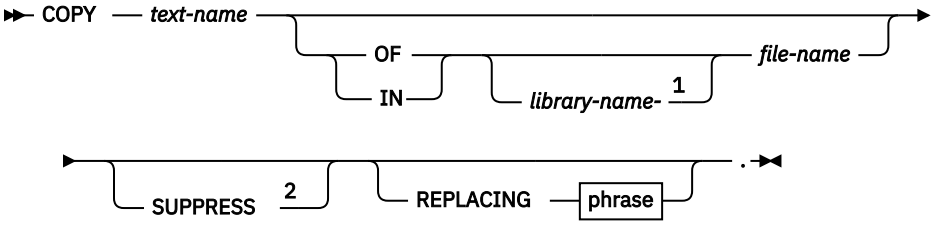
[End of IBM Extension]

Prewritten source program entries can be included in a source program at compilation 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.

COPY Statement - Format 1 - Basic



COPY Statement - Format 1

Notes:

[1] Required hyphen between library-name-file-name to qualify.

[2] IBM Extension

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.

Text-name can be qualified by the library-name and file-name in which it resides. If no file-name is specified, QCBLLESRC is assumed as the file-name. If the copy member is not found in default file QCBLLESRC then file QLBLSRC is searched next for the copy member. If the file-name is not qualified by a library-name, file-name is assumed to reside in a library in the library list. A hyphen is required between library-name and file-name, with no intervening spaces. For example, to qualify file MYFILE in library MYLIB code as MYLIB-MYFILE.

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. To qualify file-name, a hyphen is required between library-name and file-name with no intervening spaces.

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.

[IBM Extension]

COPY statements can be nested. However, nested COPY statements cannot contain the REPLACING phrase, and a COPY statement with the REPLACING phrase cannot contain nested COPY statements.

A COPY statement may not cause recursion. That is, a COPY member may be named only once in a set of nested COPY statements until the end-of-file for that COPY member is reached.

[End of IBM Extension]

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

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 511).

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.

For a nested COPY statement, SUPPRESS is in effect only until the next COPY is encountered. Suppression resumes after the nested COPY is completed.

[End of IBM Extension]

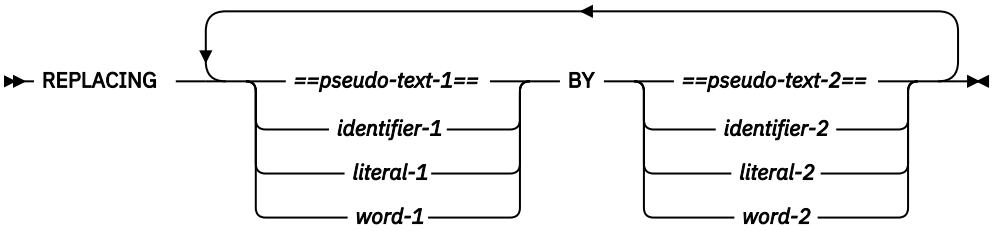
REPLACING Phrase

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)
- Function-identifier

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.

REPLACING Phrase - Format



pseudo-text-1, pseudo-text-2

Pseudo-text is a sequence of text words, comment lines, or separator spaces bounded by, but not including, the pseudo-text delimiter (==).

Pseudo-text-1 must contain at least one text word other than a separator comma or separator semicolon. Beginning and ending spaces are not included in the text comparison process, and multiple embedded spaces are considered to be a single space.

Pseudo-text-2 does not need to contain a text word; it may consist solely of space characters and/or comment lines.

For example, if library-member TEXTA consists of the following entries:

```
01  AA-DATA.
  10  AA-ID      PIC X(9).
  10  AA-TYPE    PIC X(1).
```

...the programmer can use the COPY statement to replace pseudo-text as follows:

```
    COPY TEXTA REPLACING ==01 AA-DATA== BY ==05 EE-DATA==.
                         ==AA-ID==      BY ==EE-ID==.
                         ==AA-TYPE==    BY ==EE-TYPE==.
```

...and the resulting text is treated as if it had been written as follows:

```
05  EE-DATA.
  10  EE-ID      PIC X(9).
  10  EE-TYPE    PIC X(1).
```

Pseudo-text-1 must contain, as a minimum, a text word. Since text words, by definition, are bounded by separators, pseudo-text cannot be used to select part of a word for replacement (for example, a prefix in a data name): a complete text word must be used in order to find a match. It is possible, however, to simulate the partial replacement of a text word held in the library text, by dividing it into two or more text words using separators that are used only for matching purposes, and not copied into the source program. For an example of this method, refer to “Coding Examples” on page 512.

[IBM Extension] Pseudo-text-1 or pseudo-text-2 can contain DBCS or national character-strings. Such pseudo-text cannot be continued across lines. [End of IBM Extension]

identifier-1, identifier-2

Can be defined in any Data Division section.

Can be a function-identifier.

literal-1, literal-2
Can be numeric or nonnumeric.

[IBM Extension] Can be a floating-point literal, DBCS literal, or national literal. [End of IBM Extension]

word-1, word-2
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 that do not occur as part of an identifier 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 keyword 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. For more information about the reference format areas, refer to “Reference Format” on page 41.

    Each text word copied unaltered from the library will start in the same area of the line in the resultant program as it was within the library. As an exception to this rule, however, if a text word that is being copied unaltered starts in Area A within the library, and follows another text word in Area A which is being replaced by text of a greater length, the unaltered text word will begin in Area B if it will no longer fit in Area A.

    Each text word in pseudo-text-2 that is to be placed in the resultant program begins in the same area of the resultant program as it appeared in pseudo-text-2. Each identifier-2, literal-2, and word-2 that

is to be placed in the resultant program begins in the same area of the resultant program as the library text that is being replaced.

12. [IBM Extension] COPY REPLACING does not affect the EJECT, SKIP1/2/3, or TITLE compiler-directing statements. [End of IBM Extension]

Coding Examples

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.
```

The library text will be copied unchanged into the COBOL program, immediately after the COPY statement.

Example 2: To change some (or all) of the data names within the library text used in Example 1, the programmer can use the REPLACING phrase:

```
   COPY PAYLIB REPLACING  A BY PAYROLL
                          B BY PAY-CODE
                          C BY GROSS-PAY
                          D BY HOURS.
```

When the library text is copied, the resulting text appears 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.

Example 3: This example illustrates how part of a data-name can be replaced if certain conventions are followed when creating the library text. In this case, the library text CONTACT contains the following code:

```
01 (PRFX)-RECORD.
 03 (PRFX)-NAME PIC X(24).
 03 (PRFX)-PHONE PIC X(10).
 03 (PRFX)-EXTN PIC X(4).
```

The programmer can copy this library text, replacing the text word PRFX and its bounding parentheses by a prefix for the data-names. For example, the following COPY statement can be written in the Data Division of a program:

```
    COPY CONTACT REPLACING ==(PRFX)== BY ==CUST==.
```

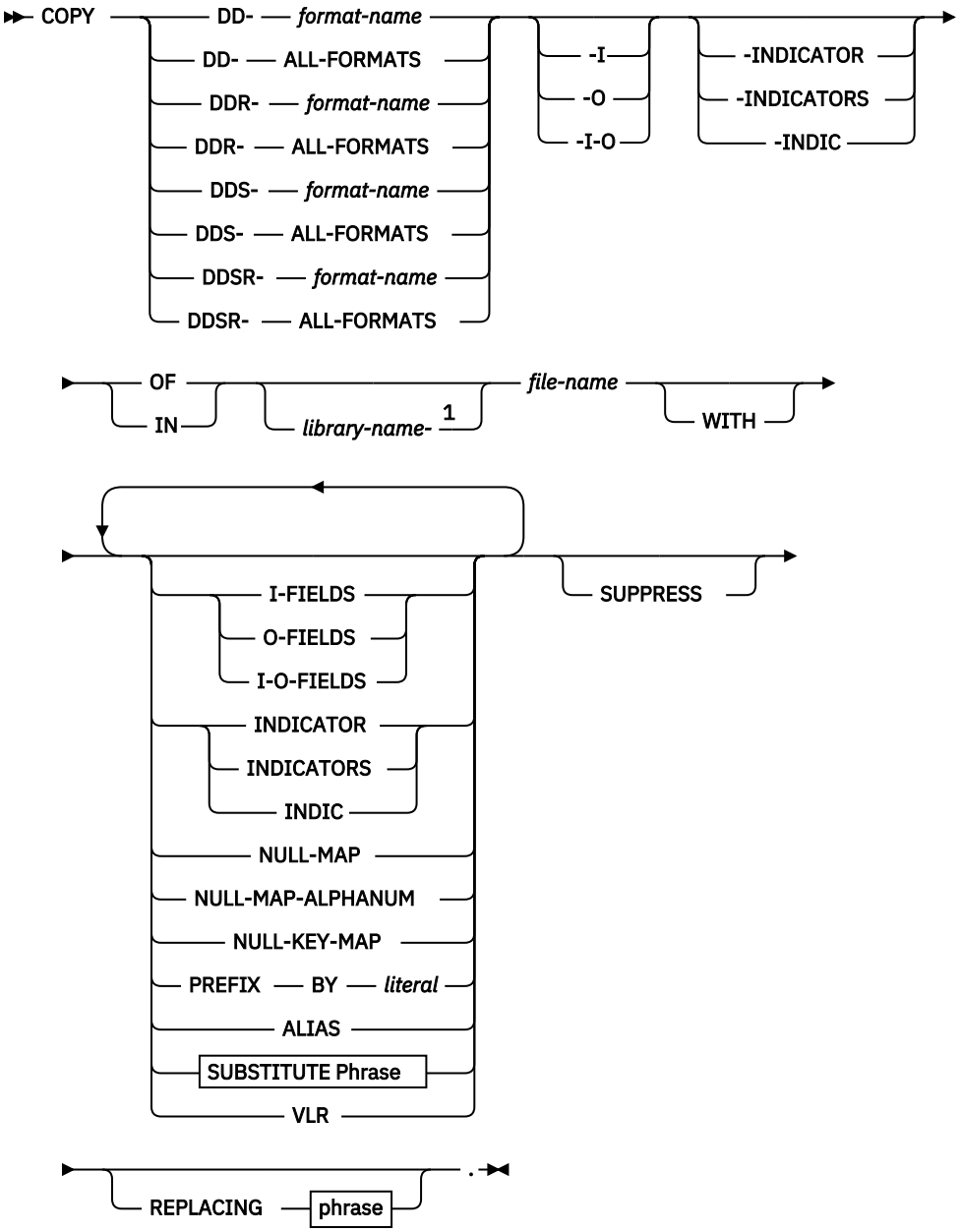
When the library text is copied, the resulting text appears as if it had been written as follows:

```
01 CUST-RECORD.
 03 CUST-NAME PIC X(24).
```

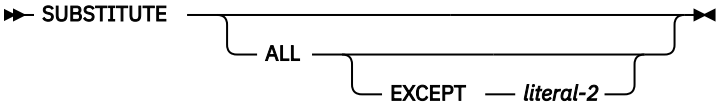
```
  03 CUST-PHONE PIC X(10).
  03 CUST-EXTN PIC X(4).
```

Note: Because many of the separators have special significance when processing a COPY statement, the values that can be used for delimiting part of a text word in this way are limited to the parenthesis and colon symbols. In addition, it will be necessary to ignore certain errors flagged by the SEU Syntax Checker when entering the library text.

[IBM Extension] COPY Statement - Format 2 - DDS Translate



SUBSTITUTE Phrase



COPY Statement - Format 2 - DDS Translate
Notes:

[1] Required hyphen between library-name-file-name to qualify.

[End of 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 compilation time. They do not make use of the DDS source statements for the file.

If a REPLACE statement is in effect, the COPY statement must be the first item on a line of code. This line must also include the text word that specifies the required options, up to at least the initial hyphen.

DDS supports DBCS with formats J (for fields which can contain only DBCS data), E (for fields which can contain either DBCS or alphanumeric data), or O (for fields can contain both DBCS and alphanumeric data). DDS also supports graphic data types with format G. The *PICGGRAPHIC option is used to create COBOL DBCS data items corresponding to format G DDS items. The *PICNGRAPHIC option is used to create COBOL NATIONAL data items with the UCS-2 CCSID specified in the National CCSID compiler option or in the NTLCCSID PROCESS option. All other circumstances produce alphanumeric data items capable of holding the correct number of bytes of data.

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 and ALIAS Options

The DD option or the ALIAS 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 or the ALIAS 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

The DDR option or the SUBSTITUTE 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

The DDS option copies in the internal DDS field names for the specified DDS format.

The DDSR Option

The DDSR option does everything that the DDS option does. It also copies the internal DDS field names in the specified DDS format, 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 and ALL-FORMATS options

The format-name is the name of the DDS record format definition that is to be translated into an ILE COBOL data description entry. The format-name must follow the rules for the formation of an ILE COBOL data-name.

The ALL-FORMATS option will translate all the formats defined for a file, including names that do not conform to the data-names rules. A REPLACING phrase must be used to change any such format-name into a valid data-name. However, a REPLACING phrase cannot be used to change a format-name within an FD entry for an indexed file defined with EXTERNALLY-DESCRIBED-KEY. If the key cannot be defined using a data-name in the RECORD KEY clause, then it will be necessary to change the format-name in the DDS specifications for the file.

Note: In this context, the compiler accepts ALL-FORMAT as the equivalent of ALL-FORMATS.

The VLR Option

The VLR option should be used with variable record files. The option specifies copying from variable-length fields. This overrides the CVTOPT(*VARCHAR) option on the CRTCBLMOD and CRTBNDCBL commands.

The PREFIX Options

The PREFIX options allows you to specify a prefix (*literal*) to be inserted in front of every field name. You can use it to help identify (that is, document) the contents or usage of the field. The literal can be contained within a pair of apostrophes or a pair of quotation marks. The maximum length of the literal allowed is 15 characters.

I-O

If neither -I or I-FIELDS, nor -O or O-FIELDS is specified, then -I-O or I-O-FIELDS is assumed. If -I and O-FIELDS, or -O and I-FIELDS is specified, then -I-O or I-O-FIELDS 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 521.

If the file is a database file, a single I-O format is generated.

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.

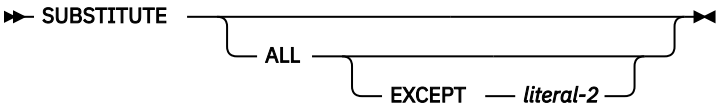
The use of the Indicator attribute is discussed under "INDICATOR Attribute of the Format 2 COPY Statement" on page 521.

File-name is the name of a 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.

SUBSTITUTE Phrase

SUBSTITUTE Phrase - Format



The SUBSTITUTE phrase allows you to bring DDS into your program, while preserving certain characters, such as the underscore character. The underscore is not a standard ILE COBOL character, but it is required for specifying locale categories. To preserve the underscore character, for example, in copied DDS, the SUBSTITUTE phrase would be used as follows:

```
      …SUBSTITUTE ALL EXCEPT '_'.
```

literal-2
Should be a 1-byte non-numeric literal. The character specified in literal-2 is not substituted.

REPLACING Phrase

The REPLACING phrase is described in "REPLACING Phrase" on page 509.

Using Null-Capable Fields in DDS Files

When a field is defined as ALWNULL in DDS, the COPY DDS statement identifies the field as null-capable with a comment. For example, the following two figures show the DDS file containing the null-capable field, and the resulting comment that is created for that field when it is copied into the ILE COBOL program's FILE-SECTION.

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
     A* With the following physical file (TESTPF)
               R TESTING
                 FLD1          5S 0
                 FLD2          8       ALWNULL
                 FLD3          6
```

Figure 29. DDS Showing Null-Capable Fields

```
* A COPY DDS-TESTING OF TESTPF.
*    I-O FORMAT:TESTING    FROM FILE TESTPF    OF LIBRARY QTEMP
*
     05  TESTING.
         06 FLD1                              PIC 9(5).
         06 FLD2                              PIC X(8).
                    (null-capable field)
         06 FLD3                              PIC X(6).
```

Figure 30. Result After Null-Capable DDS File Copied into ILE COBOL Program

To generate the null-map and null-key-map for the DDS null-capable record formats that are being copied in, the WITH NULL-MAP and WITH NULL-KEY-MAP phrases need to be specified on a new COPY DDS statement in the WORKING-STORAGE or LOCAL-STORAGE sections. Only one copy of the NULL-MAP is generated per format in the DDS. For example, if the format contains both I (input only) and B (input and output) fields, the size of the null-map generated is for all fields specified in the format. In other words, it would include all I and B fields.

For each of the null-capable fields defined in the DDS for a specific format, a data item definition is generated. The data item generated, depends on whether you specify NULL-MAP or NULL-MAP-ALPHANUM on the COPY DDS statement in the WORKING-STORAGE or LOCAL-STORAGE sections.

If you specify NULL-MAP, a null-map is created with PIC 1 values that are initialized to binary zero (0). The following statement is generated in the source for a null-capable field:

```
     06  <field-name>-NF     PIC 1  VALUE B"0".
```

If the field is not null-capable, a FILLER item is generated.

If you specify NULL-MAP-ALPHANUM, a null-map is created with PIC X values that are initialized to the character zero (0). The following statement is generated in the source for a null-capable field:

```
     06  <field-name>-NF     PIC X  VALUE ZERO.
```

If the field is not null-capable, the following statement is generated in the source:

```
     06  <field-name>-AN     PIC X  VALUE ZERO.
```

The size of a null-map generated using NULL-MAP-ALPHANUM is the same as the size of a null-map generated using NULL-MAP.

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
     A* Physical file for DDS
                R REC
                  FLD1          1A
                  FLD2          1A     ALWNULL
                  FLD3          1A
```

Figure 31. DDS File With Some Fields Not Null-Capable

```
*DDS Generated
  05 REC-NM
     06 FILLER          PIC X VALUE ZERO.
     06 FLD2-NF         PIC 1 VALUE B"0".
     06 FILLER          PIC X VALUE ZERO.
```

Figure 32. ILE COBOL Code Generated From COPY DDS with NULL-MAP

Considerations for Using Null-Capable Fields

It is possible that a null-map field can contain a value other than 1 or 0. For example, it is possible that SQL placed a value of 2 in a null-map field to indicate that the field contains a result of a divide by zero.

To be able to see a value other than 0 or 1 in a null-map, you must specify NULL-MAP-ALPHANUM on your COPY DDS statement.

NULL-MAP-ALPHANUM extends the range of values that can be received into or sent from the null map to include values other than 0 or 1. Only a value of 1 in a null map field indicates that the field is null. For more information on values other than 0 or 1 that can be sent or received in the null map, refer to the *Db2 for i* section of the *Database and File Systems* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

Using COPY DDS with Date Data Types

Table 40 on page 517 and Table 41 on page 518 list the DATFMT parameters allowed for zoned, packed, and character DDS fields, and their equivalent ILE COBOL format that is generated from COPY DDS when the CVTOPT(*CVTTODATE) conversion parameter is specified.

Table 40 on page 517 lists the IBM i DDS date data types and their equivalent ILE COBOL format. Table 40 on page 517 is for character and zoned fields; USAGE DISPLAY is assumed.

Table 40. DDS Date Data Types and Their Equivalent ILE COBOL Format

IBM i Format	COBOL-Generated Format	Description	Format	Valid Separators	Length
*MDY	%m/%d/%y	Month/Day/Year	mm/dd/yy	/-.,&	8
*DMY	%d/%m/%y	Day/Month/Year	dd/mm/yy	/-.,&	8
*YMD	%y/%m/%d	Year/Month/Day	yy/mm/dd	/-.,&	8
*JUL	%y/%j	Julian	yy/ddd	/-.,&	6
*ISO	@Y-%m-%d	International Standards Organization	yyyy-mm-dd	-	10
*USA	%m/%d/@Y	IBM USA Standard	mm/dd/yyyy	/	10
*EUR	%d.%m.@Y	IBM European Standard	dd.mm.yyyy	.	10

Table 40. DDS Date Data Types and Their Equivalent ILE COBOL Format (continued)					
IBM i Format	**COBOL-Generated Format**	**Description**	**Format**	**Valid Separators**	**Length**
*JIS	@Y-%m-%d	Japanese Industrial Standard Christian Era	yyyy-mm-dd	-	10

Table 11 on page 161 liststhe IBM i DDS time data types and their equivalent ILE COBOL format. Table 41 on page 518 is for packed fields; USAGE PACKED-DECIMAL is generated.

Table 41. DDS Time Data Types and Their Equivalent ILE COBOL Format					
IBM i Format	**COBOL-Generated Format**	**Description**	**Format**	**Valid Separators**	**Length**
*HMS	%H:%M:%S	Hours:Minutes:Seconds	hh:mm:ss	:.,&	8
*ISO	%H.%M.%S	International Standards Organization	hh.mm.ss	.	8
*USA	%I:%M @p	IBM USA Standard. AM and PM can be any mix of upper and lower case.	hh:mm AM or hh:mm PM	:	8
*EUR	%H.%M.%S	IBM European Standard	hh.mm.ss	.	8
*JIS	%H:%M:%S	Japanese Industrial Standard Christian Era	hh:mm:ss	:	8

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 or LOCAL-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
        QGPL-CUSTMASTER.   (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.)
```

Note:

1. This COPY statement generates only one storage area for all formats.

2. These COPY statements generate separate storage areas.

Data Structures Generated

This section describes the data structures generated by the COPY statement:

- FORMAT (Record) Level Structures
- Data Field Structures
- Indicator Structures

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 NAME RETRIEVAL ALTSEQ	key field number key field name ASCENDING, DESCENDING 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 in following table.)

Note: See Table 42 on page 519 for the appropriate COBOL definition.

Table 42. 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

Table 42. Data Field Structures (continued)

DDS		COBOL DATA DIVISION n=total field length (DDS pos. 30-34) m=number of decimals (DDS pos. 36 & 37)	
PHYSICAL, LOGICAL, PRINTER, AND COMMUNICATIONS FILES			
b (Blank)	Default	PIC X(n)[3]	PIC S9(n-m)V9(m)
P	Packed decimal[5]	PIC S9(n) COMP-3	PIC S9(n-m)V9(m) COMP-3
S	Zoned decimal/signed numeric[4]	PIC S9	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]	PIC 9(5) COMP-4 or COMP-1	PIC 9(5) COMP-4 or COMP-1
	Single Precision	PIC 9(10) COMP-4 or COMP-2	PIC 9(10) COMP-4 or COMP-2
	Double Precision	PIC X(n)[3]	
A	Character[4]	PIC X(n)	
H	Hexadecimal data	PIC X(n) or FORMAT DATE	
L	Date[2]	PIC X(n) or FORMAT TIME	
T	Time[2]	PIC X(n) or FORMAT TIMESTAMP	
Z	Timestamp[2]	PIC X(n)	
E	DBCS-Either data	PIC X(n)	
J	DBCS-Only data	PIC X(n)	
O	DBCS-Open data	PIC X(2n) or PIC G(n)[3]	
G	DBCS-Graphic data	—	
	UCS2-Graphic data	PIC N(2n)	
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]	PIC 9(5) COMP-4 or COMP-1	PIC 9(5) COMP-4 or COMP-1
	single precision	PIC 9(10) COMP-4 or COMP-2	PIC 9(10) COMP-4 or COMP-2
	double precision	PIC X(n)	—
M	Numeric-only Character	PIC X(n) or FORMAT DATE	—
L	Date[2]	PIC X(n)or FORMAT TIME	—
T	Time[2]	PIC X(n) or FORMAT TIMESTAMP	—
Z	Timestamp[2]	—	PIC S9(n-m)V9(m)
S	Signed-Numeric Shift	PIC X(n)	—
E	DBCS-either	PIC X(n)	—
J	DBCS-only	PIC X(n)	—
O	DBCS-open	PIC X(2n) or PIC G(n)	—
G	DBCS-graphic	—	
	UCS2-graphic	PIC N(2n)	

Note:

1. If the *NOFLOAT value of the CVTOPT parameter is in effect, then floating-point fields are brought in as FILLER items with USAGE of BINARY. If *FLOAT is specified, the fields are brought in using their given DDS names with a USAGE of COMP-1 (single-precision floating-point) or USAGE of COMP-2 (double-precision floating-point). See "Floating-Point Fields" on page 523.
2. FILLER items are declared as alphanumeric by default. You can also have COBOL treat date, time, and timestamp fields as date-time data types by specifying *DATE, *TIME, or *TIMESTAMP on the CVTOPT parameter of the CRTBNDCBL or CRTCBLMOD command. See "Date, Time, and Timestamp Fields" on page 523.
3. In DDS, if the field has an attribute of VARLEN, the result is two additional bytes at the beginning of the field.
4. If you have a DDS character or zoned data type with the DATFMT keyword, ILE COBOL treats it as a date field, if the *CVTTODATE value of the CVTOPT parameter in the CRTBNDCBL or CRTCBLMOD command is specified.
5. If you have a DDS packed data type with the DATFMT keyword, ILE COBOL treats it as a date field, if the *CVTTODATE value of the CVTOPT parameter in the CRTBNDCBL or CRTCBLMOD command is specified.
6. In DDS, if a field with data type G has an attribute CCSID(n), where n is the CCSID specified in the National CCSID compiler option or in the NTLCCSID PROCESS option, it is a UCS-2 graphic data type. To bring in the UCS-2 graphic data type as a COBOL NATIONAL data type, specify *PICNGRAPHIC on the CVTOPT parameter of the CRTBNDCBL or CRTCBLMOD command. For more information, see *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

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” on page 521.

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.(See Figure 33 on page 522.)

```
5722WDS V5R4M0  060210 LN  IBM ILE COBOL         TESTLIB/STRTEXTD     I-SERIES1  06/02/15 11:27:50     Page  2
                                 S o u r c e
STMT PL SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S COPYNAME   CHG DATE
  1     000100 IDENTIFICATION DIVISION.
  2     000200 PROGRAM-ID. STRTEXTD.
        000300
  3     000400 ENVIRONMENT DIVISION.
  4     000500 CONFIGURATION SECTION.
  5     000600   SOURCE-COMPUTER. IBM-ISERIES.                                                    02/02/21
  6     000700   OBJECT-COMPUTER. IBM-ISERIES.                                                    02/02/21
  7     000800 INPUT-OUTPUT SECTION.                                                             02/02/21
  8     000900 FILE-CONTROL.
  9     001000     SELECT FILE-1 ASSIGN TO DATABASE-NAMES
 11     001100     ACCESS IS DYNAMIC RECORD KEY IS EXTERNALLY-DESCRIBED-KEY
 13     001200     ORGANIZATION IS INDEXED.
        001300
 14     001400 DATA DIVISION.
 15     001500 FILE SECTION.
 16     001600 FD  FILE-1.
 17     001700 01  RECORD-DESCRIPTION.                                                           02/02/21
        001800     COPY DDS-RDE OF NAMES.                                                       02/02/21
       +000001*    I-O FORMAT:RDE        FROM FILE NAMES      OF LIBRARY TESTLIB            RDE
       +000002*                          RECORD DESCRIPTION                                 RDE
       +000003*THE KEY DEFINITIONS FOR RECORD FORMAT  RDE                                    RDE
       +000004*  NUMBER                NAME                RETRIEVAL       ALTSEQ         RDE
       +000005*   0001   LNAME                             ASCENDING         NO             RDE
       +000006*   0002   FNAME                             ASCENDING         NO             RDE
       +000007*   0003   MINAME                            ASCENDING         NO             RDE
       +000008*   0004   MNAME                             ASCENDING         NO             RDE
 18    +000009       05  RDE.                                                                RDE
 19    +000010           06 FNAME                PIC X(20).                                  RDE
       +000011*                FIRST NAME                                                    RDE
 20    +000012           06 MINAME               PIC X(1).                                   RDE
       +000013*                MIDDLE INITIAL NAME                                           RDE
 21    +000014           06 MNAME                PIC X(19).                                  RDE
       +000015*                REST OF MIDDLE NAME                                           RDE
 22    +000016           06 LNAME                PIC X(20).                                  RDE
       +000017*                LAST NAME                                                     RDE
 23    +000018           06 PHONE                PIC S9(10)      COMP-3.                     RDE
       +000019*                PHONE NUMBER                                                  RDE
 24    +000020           06 DATA-DDS             PIC X(40).                                  RDE
       +000021*                REST OF DATA                                                  RDE
 25     001900 66  MIDDLE-NAME RENAMES MINAME THRU MNAME.
        002000
 26     002100 PROCEDURE DIVISION.
        002200 MAIN-PROGRAM SECTION.
        002300 MAINLINE.
 27     002400     OPEN INPUT FILE-1.
        002500*      .
        002600*      .
        002700*      .
```

Figure 33. Generation of I-O Formats

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 or Local-Storage sections 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, Working-Storage and Local-Storage Sections, but the results are not exactly the same. For more information, see “Key Generation Examples” on page 526.

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 the *IBM Rational Development Studio for i: ILE COBOL Programmer'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

A file can contain internal floating-point fields. If the *NOFLOAT value of the CVTOPT parameter (the default) is in effect, then the floating-point fields are brought in as FILLER items with a USAGE of BINARY. If *FLOAT is specified, the fields are brought in using their given DDS names with a USAGE of COMP-1 (single precision floating-point) or COMP-2 (double precision floating-point).

Floating-point key fields are allowed. If the KEY is an internal floating-point number, the sequence of key values will be in numeric order. If the KEY is an external floating-point number, the key is alphanumeric, and the sequence of the records depends on the collating sequence used.

Date, Time, and Timestamp Fields

This section describes the following classes of date, time, and timestamp fields:

- Class Date-Time
- Class Alphanumeric

It also provides an example of how use DDS to define date, time, and timestamp fields of class date-time.

Class Date-Time

Date-time fields include date, time, and timestamp data items of class date-time, and are allowed for zoned, packed, and character DDS fields that specify the DATFMT keyword. Date data types are not the

same as date, time, and timestamp fields that are brought into your program as fixed-length character fields. In ILE COBOL, date data types are converted to USAGE DISPLAY or USAGE PACKED-DECIMAL data items, and date, time, and timestamp fields are converted to alphanumeric data items (as described in “Class Alphanumeric” on page 524).

Date data types are converted to their equivalent ILE COBOL format from COPY DDS when the *CVTTODATE conversion parameter option (CVTOPT) is specified.

For more information about the DATFMT parameters allowed and the equivalent ILE COBOL format that is generated for them from COPY DDS when the CVTOPT(*CVTTODATE) conversion parameter is specified, refer to the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

Class Alphanumeric

Alphanumeric date, time, and timestamp fields are brought into your program only if you specify the CVTOPT(*DATETIME) option on the CRTCBLMOD or CRTBNDCBL command, or the DATETIME option of the PROCESS statement. If *DATETIME is not specified, date, time, and timestamp fields are ignored and are declared as FILLER fields in your ILE COBOL program.

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 *Db2 for i* section of the *Database and File Systems* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

Example of Date, Time, and Timestamp DDS

The following example shows you how to define date, time, and timestamp fields in DDS.

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
     A               LOGICAL FILE LF1 FOR DATE, TIME, AND TIMESTAMP EXAMPLES
00010A
00020A          R RECORD1
00030A            DATFLD1         L         DATFMT(*JUL)
00040A                                      ALIAS(A_DATE_JUL)
00050A            DATFLD2         L         DATFMT(*EUR)
00060A                                      ALIAS(A_DATE_EUR)
00070A            DATFLD3         L         DATFMT(*DMY) DATSEP('-')
00080A                                      ALIAS(A_DATE_DMY)
00090A            DATFLD4         L         DATSEP(' ')
00100A            TIMFLD1         T         TIMFMT(*ISO)
00110A                                      ALIAS(A_DATE_ISO)
00120A            TIMFLD2         T         TIMFMT(*USA)
00130A                                      ALIAS(A_DATE_USA)
00140A            TIMFLD3         T         TIMSEP(' ')
00150A            TIMFLD4         T         TIMSEP('.')
00160A            TSFLD1          Z         DFT('1998-02-27-08.15.22.000000')
     A
```

If the current date is June 21, 1990, the current system date format value is MDY, and the system date separator value is '/', DATFLD3 contains 21-06-90. DATFLD4 contains 06 21 90.

If the current date is June 21, 1990, the current system date format value is MDY, and the current system separator is /, DATFLD1 contains 90/172 (the 172nd day of the year 1990). DATFLD2 contains 21.06.1990.

If the current time is 2 o'clock p.m., the system time format is hhmmss, and the system time separator is ':', TIMFLD1 contains 14.00.00. TIMFLD2 contains 2:00 PM.

If the current time is 2 o'clock p.m., the system time format is hhmmss, and the system time separator is ':', TIMFLD3 contains 14 00 00. TIMFLD4 contains 14.00.00.

If you are defining a timestamp field, you must specify the default value in the following format:

```
    DFT('YYYY-MM-DD-HH.MM.SS.UUUUUU')
```

If the DFT keyword is not specified, the default value is the current time.

Figure 34. DDS File With Date, Time, and Timestamp Fields Defined

Variable-Length Fields

You can bring a variable-length field into your program if you specify the CVTOPT(*VARCHAR) option on the CRTCBLMOD or CRTBNDCBL command, 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 *IBM Rational Development Studio for i: ILE COBOL Programmer'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 509 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.

Figure 35 on page 526 describes the Format 2 COPY statement **without** the REPLACING option:

```
5722WDS V5R4M0  060210 LN  IBM ILE COBOL  TESTLIB/STRTEXTD   I-SERIES1  06/02/15 11:27:50   Page   2
STMT PL SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN  S COPYNAME  CHG DATE

 11    000100 FD  CUST-MASTER.
 12    000200 01  CUSTOMER-RECORD.
       000300*
       000400* COPY DDS  W I T H O U T  REPLACING OPTION
       000500*
 13    000600     COPY DDS-CUSMST OF TESTLIB-CUSMSTP.
       +000001*    I-O FORMAT:CUSMST     FROM FILE CUSMSTP    OF LIBRARY TESTLIB           CUSMST
       +000002*                          ORDER HEADER RECORD                               CUSMST
 14    +000003       05  CUSMST.                                                           CUSMST
 15    +000004           06 CUST              PIC X(5).                                    CUSMST
       +000005*                          CUSTOMER NUMBER                                   CUSMST
 16    +000006           06 NAME              PIC X(25).                                   CUSMST
       +000007*                          CUSTOMER NAME                                     CUSMST
 17    +000008           06 ADDR              PIC X(20).                                   CUSMST
       +000009*                          CUSTOMER ADDRESS                                  CUSMST
 18    +000010           06 CITY              PIC X(20).                                   CUSMST
       +000011*                          CUSTOMER CITY                                     CUSMST
 19    +000012           06 STATE             PIC X(2).                                    CUSMST
       +000013*                          STATE                                             CUSMST
 20    +000014           06 ZIP               PIC S9(5)        COMP-3.                     CUSMST
       +000015*                          ZIP CODE                                          CUSMST
```

Figure 35. Format 2 COPY Statement Without the REPLACING Option

The following figure describes the Format 2 COPY Statement **with** the REPLACING option:

```
5722WDS V5R4M0  060210 LN  IBM ILE COBOL     TESTLIB/STRTEXTD     I-SERIES1  06/02/15 11:27:50   Page   2
STMT PL SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN  S COPYNAME   CHG DATE

 30    001000 FD  CUST-MASTER.
 31    001100 01  CUSTOMER-RECORD.
       001200*
       001300* COPY DDS  W I T H  REPLACING OPTION
       001400*
 32    001500     COPY DDS-CUSMST OF TESTLIB-CUSMSTP
 33    001600          REPLACING NAME BY ADDR-LINE-1
 34    001700                    ADDR BY ADDR-LINE-2
 35    001800                    CITY BY ADDR-LINE-3.
       +000001*    I-O FORMAT:CUSMST     FROM FILE CUSMSTP    OF LIBRARY TESTLIB           CUSMST
       +000002*                          ORDER HEADER RECORD                               CUSMST
 36    +000003       05  CUSMST.                                                           CUSMST
 37    +000004           06 CUST              PIC X(5).                                    CUSMST
       +000005*                          CUSTOMER NUMBER                                   CUSMST
 38    +000006           06 ADDR-LINE-1           PIC X(25).                              CUSMST
       +000007*                          CUSTOMER NAME                                     CUSMST
 39    +000008           06 ADDR-LINE-2           PIC X(20).                              CUSMST
       +000009*                          CUSTOMER ADDRESS                                  CUSMST
 40    +000010           06 ADDR-LINE-3           PIC X(20).                              CUSMST
       +000011*                          CUSTOMER CITY                                     CUSMST
 41    +000012           06 STATE             PIC X(2).                                    CUSMST
       +000013*                          STATE                                             CUSMST
 42    +000014           06 ZIP               PIC S9(5)        COMP-3.                     CUSMST
       +000015*                          ZIP CODE                                          CUSMST
```

Figure 36. Format 2 COPY Statement With the REPLACING Option

Key Generation Examples

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A              PHYSICAL FILE PF1 FOR KEY GENERATION EXAMPLES
     A
     A          R PFRECORD
     A
     A            MTH            2
     A            DAY            2
     A            YEAR           4
     A            ITEM           5
     A
     A
     A          K MTH
     A          K DAY
```

Figure 37. Data Description Specifications for a Physical File

The physical file described by Figure 37 on page 527 forms a basis for the examples that follow. Each example refers to a logical file (derived from the physical file) that specifies EXTERNALLY-DESCRIBED-KEY in its SELECT clause.

Example Using CONCAT Keyword

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A              LOGICAL FILE LF1 FOR CONCAT KEYWORD EXAMPLES
     A
     A          R RECORD1                  PFILE(PF1)
     A
     A            DATE                     CONCAT(MTH DAY YEAR)
     A
     A          K MTH
     A          K DAY
```

Figure 38. Data Description Specifications Using the CONCAT Keyword

For the logical file described by Figure 38 on page 527, COPY DDS generates keys and key names derived from the physical file.

```
    FD LF1 LABEL RECORDS ARE STANDARD.
    01  LOG-RECORD.
                COPY DDS-ALL-FORMATS OF LF1.
        05  LF1-RECORD PIC X(8).
   *  I-O FORMAT:RECORD-1   FROM FILE LF1        OF LIBRARY COPYDDS
   *
   *THE KEY DEFINITIONS FOR RECORD FORMAT  RECORD1
   *  NUMBER              NAME               RETRIEVAL    TYPE   ALTSEQ
   *   0001  MTH-DDS                          ASCENDING     AN     NO
   *         KEY NAME ORIGINATES FROM PHYSICAL FILE
   *   0002  DAY-DDS-DDS                      ASCENDING     AN     NO
   *         KEY NAME ORIGINATES FROM PHYSICAL FILE
        05 RECORD1       REDEFINES LF1-RECORD.
           06 DATE-DDS              PIC X(8).
           06 FILLER REDEFINES DATE-DDS.
              07 MTH-DDS            PIC X(2).
              07 DAY-DDS-DDS        PIC X(2).
              07 FILLER             PIC X(4).
```

Figure 39. Example Using the CONCAT Keyword

The COPY statement adds the suffix -DDS to the field names MTH and DATE because MTH is a key that originates from the physical file, and DATE is an ILE COBOL reserved word. The COPY statement adds the suffix -DDS twice to the field name DAY because DAY is both a key that originates from the physical file and an ILE COBOL reserved word.

Note that if you move your COPY statement from the File Section to the Working-Storage Section or to the Linkage Section, the fields subordinate to DATE-DDS are no longer available:

```
    WORKING-STORAGE SECTION.
    01  WRK-RECORD.
                COPY DDS-ALL-FORMATS OF LF1.
       05  LF1-RECORD PIC X(8).
   *  I-O FORMAT:RECORD-1   FROM FILE LF1         OF LIBRARY COPYDDS
   *
       05  RECORD1      REDEFINES LF1-RECORD.
           06 DATE-DDS             PIC X(8).
```

Figure 40. Example Using the CONCAT Keyword—Working-Storage Section

Example Using RENAME Keyword

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A               LOGICAL FILE LF2 FOR RENAME KEYWORD EXAMPLES
     A
     A          R RECORD2                   PFILE(PF1)
     A
     A            MONTH                     RENAME(MTH)
     A
     A          K MTH
```

Figure 41. Data Description Specifications Using the RENAMES Keyword

For the logical file described by Figure 37 on page 527, COPY DDS generates a key and key name derived from the physical file:

```
    *
     FD LF2 LABEL RECORDS ARE STANDARD.
     01  LOG-RECORD.
                 COPY DDS-ALL-FORMATS OF LF2.
       05  LF2-RECORD PIC X(2).
   *  I-O FORMAT:RECORD2    FROM FILE LF2         OF LIBRARY COPYDDS
   *
   *THE KEY DEFINITIONS FOR RECORD FORMAT  RECORD2
   *  NUMBER              NAME              RETRIEVAL    TYPE   ALTSEQ
   *   0001  MTH-DDS                         ASCENDING     AN     NO
   *         KEY NAME ORIGINATES FROM PHYSICAL FILE
       05  RECORD2      REDEFINES LF2-RECORD.
           06 MONTH                PIC X(2).
           06 MTH-DDS REDEFINES MONTH PIC X(2).
```

Figure 42. Using the RENAME Keyword

The COPY statement adds the suffix -DDS to the field name MTH because MTH is a key that originates from the physical file.

Example Using SST Keyword

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A              LOGICAL FILE LF3 FOR SST KEYWORD EXAMPLES
     A
     A          R RECORD3                   PFILE(PF1)
     A
     A            YY                I      SST(YEAR 2 2)
     A
     A          K YY
```

Figure 43. Data Description Specifications Using the SST Keyword

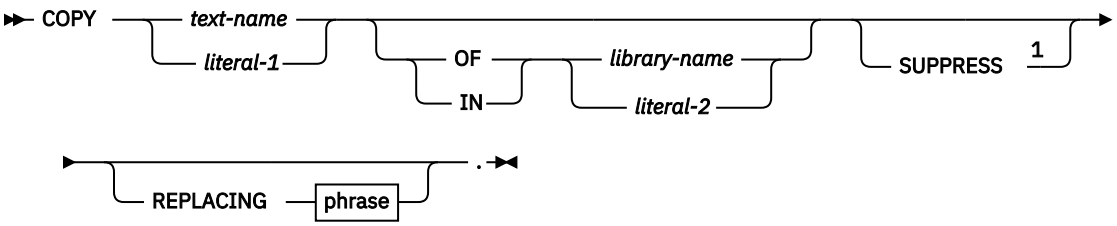
For the logical file described by Figure 37 on page 527, COPY DDS generates the following specifications:

```
    *
     FD LF3 LABEL RECORDS ARE STANDARD.
     01  LOG-RECORD.
                 COPY DDS-ALL-FORMATS OF LF3.
         05  LF3-RECORD PIC X(2).
    *  I-O FORMAT:RECORD3    FROM FILE LF3         OF LIBRARY COPYDDS
    *
    *THE KEY DEFINITIONS FOR RECORD FORMAT  RECORD3
    *  NUMBER              NAME              RETRIEVAL    TYPE   ALTSEQ
    *   0001  YY                              ASCENDING     AN     NO
        05  RECORD3      REDEFINES LF3-RECORD.
            06 YY                   PIC X(2).
```

Figure 44. Using the SST Keyword

The COPY statement does not add a suffix to the field name YY because YY is neither a key that originates from the physical file nor an ILE COBOL reserved word.

[IBM Extension] COPY Statement - Format 3 - Basic IFS



Copy Statement - Format 3 - Basic IFS

Notes:

[1] IBM Extension

literal-1 and literal-2

`literal-1` is the name of the stream file to copy. If library-name is omitted, the literal is used directly: as a file name, a relative path name, or an absolute path name (if the first character is '/'). For example:

```
COPY "MyInc"
COPY "x/MyInc"
COPY "/u/user1/MyInc"
COPY "/QSYS.LIB/QSYSINC.LIB/QCBLLESRC.FILE/JNI.MBR"
```

`literal-2` is treated as the actual path, relative or absolute, from which the copy file text-name or literal-1 is located.

text-name

When text-name is a user-defined Cobol word and an environment variable of that name is defined, the value of the environment variable is used as the name of the file containing the copy text. If an

environment variable of that name is not defined, the copy text is searched for according to the following names, in the order specified as follows:

```
1. text-name.cpy
2. text-name.CPY
3. text-name.cblle
4. text-name.CBLLE
5. text-name.cbllеinc
6. text-name.CBLLEINC
7. text-name.cbl
8. text-name.CBL
9. text-name.cob
10. text-name.COB
11. text-name.MBR
12. text-name
```

library-name

When library-name is a user-defined Cobol word, it is treated as an environment variable. The value of the environment variable is used as the path from which the copy file, text-name or literal-1 is located. If the environment variable is not set, an error occurs. If both library-name and text-name are specified, the compiler forms the path name for the copy text by concatenating library-name and text-name with a path separator (/) inserted between the two values. For example, suppose you have the following setting for COPY MYCOPY OF MYLIB:

```
MYCOPY=mystuff/today.cpy
MYLIB=/u/user1
```

These settings result in:

```
/u/user1/mystuff/today.cpy
```

When library-name is an environment variable that identifies the path from which copy text is to be copied, use the ADDENVVAR command such as the following example to define library-name:

```
ADDENVVAR ENVVAR(COPYLIB) VALUE(/u/mystuff/copybooks)
```

The name of the environment variable must be uppercase. To specify more than one copy library, set the environment variable to multiple path names delimited by : (colon). When library-name is omitted and text-name is not an absolute path name, the copy text is searched for in this order:

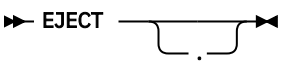
1. In the current directory
2. In the paths specified on the INCDIR parameter
3. In the paths specified in the SYSLIB environment variable

[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.

EJECT Statement - 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]

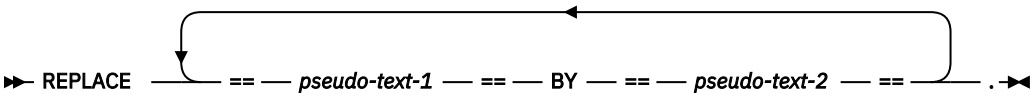
REPLACE Statement

The REPLACE statement is used to replace source program text.

The REPLACE statement can occur anywhere in the source program where a character-string can occur. It must be preceded by a separator period except when it is the first statement in a separately compiled program. It must be terminated by a separator period.

The REPLACE statement resembles the REPLACING phrase of the COPY statement, except that it acts on the entire source program, not just the text in COPY libraries.

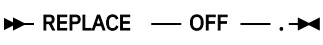
REPLACE Statement - Format 1



Each matched occurrence of pseudo-text-1 is replaced by its corresponding pseudo-text-2. This process continues until any of the following are met:

- The next occurrence of the REPLACE statement
- End of the program
- REPLACE OFF (see Format 2 below)

REPLACE Statement - Format 2



Format 2 ends current text replacement specified by Format 1.

pseudo-text-1, pseudo-text-2

Pseudo-text is a sequence of text words, comment lines, or separator spaces bounded by, but not including, the pseudo-text delimiter (==).

Pseudo-text-1 must contain at least one text word other than a separator comma or separator semicolon. Beginning and ending spaces are not included in the text comparison process, and multiple embedded spaces are considered to be a single space.

Pseudo-text-2 does not need to contain a text word; it may consist solely of space characters and/or comment lines.

Since pseudo-text-1 requires a text word, which must be bounded by separators, pseudo-text cannot be used to replace part of a data name (for example, a prefix); the entire data name must be used.

[IBM Extension] Pseudo-text-1 or pseudo-text-2 can contain DBCS or national character-strings. Such pseudo-text cannot be continued across lines. [End of IBM Extension]

[IBM Extension] When a REPLACE statement is in effect, there are certain restrictions on the layout of a Format 2 - DDS Translate COPY statement. (See “COPY Statement - Format 2 - DDS Translate” on page 513.) [End of IBM Extension]

REPLACE statements are processed after all COPY statements have been processed. The text that results from the processing of a REPLACE statement must not include a REPLACE statement.

Replacing Algorithm

For example, assuming three matched pairs of pseudo-text in a REPLACE statement:

1. The comparison starts with the leftmost source program text word following the REPLACE statement, and with the first pseudo-text-1.
2. Pseudo-text-1 is compared to an equivalent number of contiguous source program text words according to the following rules:
   - The comparison is character for character

- Uppercase and lowercase characters are equivalent (except within literals)
- Each occurrence of a separator comma, semicolon, and sequence of one or more spaces is treated as a single space
- Comment lines and blank lines are ignored for purposes of matching.
- [IBM Extension] Lines containing EJECT, SKIP 1/2/3, or TITLE statements are ignored for purposes of matching (they are treated as comment lines) [End of IBM Extension]
- Debugging lines are processed for matches, but the D in the indicator area is ignored

3. If no match occurs, the comparison is repeated with each successive occurrence of pseudo-text-1 (in our example, there are three occurrences), until a match is found (go to Step 5)
4. If no match is found after all, the next source program text word is treated as the leftmost program text word, and the cycle begins again at Step 1
5. When a match is found, the corresponding pseudo-text-2 replaces the matched text in the source program
6. The source program text word immediately following the rightmost text word that participated in the match becomes the leftmost source program text word. The cycle starts again (Step 1) with the first occurrence of pseudo-text-1.

Programming Notes

After all COPY and REPLACE statements are processed, and if the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph, a debugging line is considered to have all the characteristics of a comment line

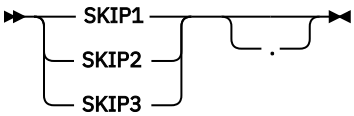
When additional lines are introduced in the source program as a result of processing the REPLACE statement, the indicator area of the new lines contains the same character as the line of the text being replaced (unless the line contains a hyphen, in which case the new lines contain a space)

If a literal within pseudo-text-2 does not fit in the line containing pseudo-text-1, and the literal is not being placed in a debugging line, additional continuation lines are introduced that contain the remainder of the literal. If pseudo-text-1 is on a debugging line, the program is in error.

[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.

SKIP1/2/3 Statements - 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.

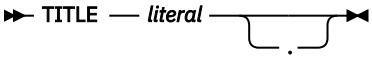
SKIP1, SKIP2, or SKIP3 may be written anywhere in either Area A or Area B, and may be terminated with a separator period. It must be the only statement on the line.

[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. The title line is printed below the line containing the identification of the compiler and the current release level. The title is left-justified on the title line.

TITLE Statement - Format



literal

Must be nonnumeric and may be followed by a separator period. Must not be a figurative constant. May be a DBCS literal or national literal.

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]

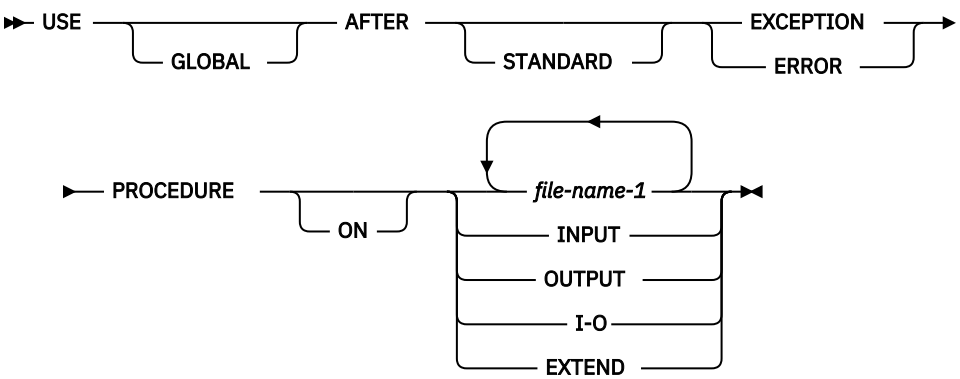
USE Statement

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. (See “Precedence Rules for Nested Programs” on page 535 for information on using the GLOBAL phrase.)

USE Statement - Format 1 - EXCEPTION/ERROR

The words EXCEPTION and ERROR are synonymous and may be used interchangeably.

USE Statement - 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. 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.

[IBM Extension] The file-name phrase is also valid for TRANSACTION files. [End of IBM Extension]

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
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 250.)

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 250.

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.

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.

Precedence Rules for Nested Programs

Special precedence rules are followed when programs are contained within other programs. In applying these rules, only the first qualifying declarative will be selected for execution. The declarative that is selected must satisfy the rules for execution of that declarative. The order of precedence for selecting a declarative is:

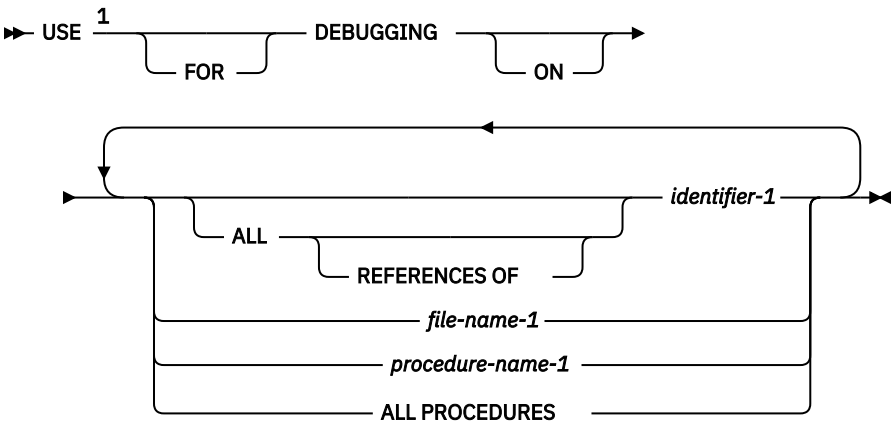
1. A file-specific declarative (one of the form *USE AFTER ERROR ON file-name-1*, with or without the GLOBAL phrase) within the program that contains the statement that caused the qualifying condition
2. A mode-specific declarative (one of the form *USE AFTER ERROR ON INPUT*, with or without the GLOBAL phrase) within the program that contains the statement that caused the qualifying condition
3. A file-specific declarative that specifies the GLOBAL phrase, and is within the program directly containing the program that was last examined for a qualifying condition
4. A mode-specific declarative that specifies the GLOBAL phrase, and is within the program directly containing the program that was last examined for a qualifying condition.
5. Rules 3 and 4 apply recursively back through the parents in the nest of programs.

Note: Each declarative procedure runs as a separate invocation from that of other declarative procedures and the nondeclarative part of the same ILE COBOL program.

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.

The USE FOR DEBUGGING declarative is syntax checked and treated as documentation.



USE FOR DEBUGGING Declarative - Format

Notes:

[1] Syntax-checked only.

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.

Chapter 9. Appendixes

Appendix A. ILE COBOL Compiler Limits

The following table lists the compiler limits supported by the ILE COBOL compiler:

Table 43. ILE COBOL Compiler Limits

Language Element	**ILE COBOL Limit**
General	
Number of: Files open at one time Nesting levels in nested COPY REPLACING operands in one COPY	 virtually no limit (1) virtually no limit (1) virtually no limit (1)
Total length of literals	virtually no limit (1)
Total storage available for VALUE clauses	virtually no limit (1)
Number of characters to identify: Library-name Program-name program object ILE procedure Text-name	 10 10 250 10
Environment Division	
Number of: SELECT file-names Alternate record keys in one file Contiguous DDS fields that can be used to form an alternate record key	 virtually no limit (1) 253 156
Maximum number of buffers (areas) specified in the RESERVE clause	 virtually no limit (1)

Table 43. ILE COBOL Compiler Limits (continued)	
Language Element	**ILE COBOL Limit**
Length of: RECORD KEY in one file ALTERNATE RECORD KEY in one file	 2 000 bytes 2 000 bytes
Data Division	
Length of: Working-Storage Section group item Linkage Section group item Local-Storage Section Elementary item	 16 711 568 bytes 16 711 568 bytes 16 711 568 bytes 16 711 568 bytes
Maximum block size	32 767 bytes
Maximum record length	32 767 bytes
Number of: FD file-names OCCURS levels Levels in data hierarchy SD file-names	 virtually no limit (1) 7 49 virtually no limit (1)
Number of: Numeric-edited (data items) character positions Picture character strings Picture replications	 127 90 16 711 568
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 Index names (per table) INDEXED BY clauses (per table) Pointers in one table	 16 711 568 bytes 16 711 568 bytes 16 711 568 bytes virtually no limit (1) virtually no limit (1) virtually no limit (1) 1 virtually no limit (1)
Procedure Division	

Table 43. ILE COBOL Compiler Limits (continued)

Language Element	ILE COBOL Limit
Number of:	
GO TO procedure-name DEPENDING ON	virtually no limit (1)
nested IF statements	virtually no limit (1)
IF nesting levels	virtually no limit (1)
nested EVALUATE statements	virtually no limit (1)
CALL parameters	
to program object	255
to ILE procedure	400
FUNCTION nesting levels limit	
for intrinsic functions	123
SORT-MERGE input files	32
SORT-MERGE output files	32
SORT-MERGE keys	2 000
SEARCH ALL ... WHEN	
relation conditions	virtually no limit (1)
UNSTRING delimiters	virtually no limit (1)
INSPECT TALLYING identifiers	virtually no limit (1)
INSPECT REPLACING identifiers	virtually no limit (1)
Length of:	
SORT-MERGE keys	2 000 bytes

Note:

1. The limit is a very large number, depending on your hardware configuration. Most applications should not encounter it.

Appendix B. Intermediate Results and Arithmetic Precision

The compiler handles arithmetic statements as a succession of operations, performed according to operator precedence, and sets up an intermediate field to contain the results of these operations.

Intermediate results are possible in the following cases:

- In an ADD or SUBTRACT statement containing more than one operand immediately following the verb.
- In a COMPUTE statement specifying a series of arithmetic operations or multiple result fields.
- In arithmetic expressions contained in conditional statements and reference modification specifications.
- In the GIVING option with multiple result fields for the ADD, SUBTRACT, MULTIPLY, or DIVIDE statements.
- In a statement with an intrinsic function used as an operand.

For a discussion on when the compiler uses fixed-point or floating-point arithmetic, refer to the "Working with Data Items" chapter in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

Calculating Precision of Intermediate Results

The compiler uses algorithms to determine the number of integer and decimal places reserved for intermediate results.

In the following discussion of how the compiler determines the number of integer and decimal places reserved for intermediate results, these abbreviations are used:

i
The number of integer places carried for an intermediate result.

d
The number of decimal places carried for an intermediate result.

ROUNDED
If the ROUNDED option is used, one more integer or decimal place might be added for accuracy, if necessary. Only the final results are rounded; the intermediate results are not rounded. 62 digits is the maximum number of digits that can be accurately rounded.

dmax
In a particular statement, the largest of:

- The number of decimal places needed for the final result field(s).
- The maximum number of decimal places defined for any operand.
- The outer-dmax for any function operand.

inner-dmax
The inner-dmax for a function is the largest of:

- The number of decimal places defined for any of its elementary arguments.
- The dmax for any of its arithmetic expression arguments.
- The outer-dmax for any of its embedded functions.

outer-dmax
The number that determines how a function result contributes to operations outside of its own evaluation (for example if the function is an operand in an arithmetic expression or an argument to another function).

op1
The first operand in a generated arithmetic statement. For division, op1 is the divisor.

op2
The second operand in a generated arithmetic statement. For division, op2 is the dividend.

i1,i2
The number of integer places in op1 and op2, respectively.

d1,d2
The number of decimal places defined for op1 and 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.

Below we use a COMPUTE statement to demonstrate the use of intermediate results in an arithmetic expression. In this case, 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:

Table 44. Determining the Precision of an Intermediate Result

Operation	Integer Places	Decimal Places
+ or -	(i1 or i2) + 1, whichever is greater	d1 or d2, whichever is greater
*	i1 + i2	d1 + d2
/	i2 + d1	(d2 - d1) or dmax, whichever is greater
**	**When i2 equals 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 does not equal 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

Note:

1. These results are subject to subsequent processing.

You must define the operands of any arithmetic statements with enough decimal places to give the desired accuracy in the final result.

Table 45 on page 542 indicates the action of the compiler when handling intermediate results for fixed-point numbers.

Table 45. Determining When the Compiler Might Truncate Intermediate Results

Value of i + d[2]	Value of d	Value of i + dmax	Action Taken
< MAXLENGTH[1] = MAXLENGTH	Any Value	Any Value	*i* integer and *d* decimal places are carried for ir.
> MAXLENGTH[3]	< dmax = dmax	Any Value	MAXLENGTH - *d* integer and *d* decimal places are carried for ir.
	> dmax	< MAXLENGTH = MAXLENGTH	*i* integer and MAXLENGTH - *i* decimal places are carried for ir.
		> MAXLENGTH	MAXLENGTH - dmax integer and dmax decimal places are carried for ir.

Note:

1. MAXLENGTH has one of the following values:
   - 18 decimal digits

   [IBM Extension]
   - 30 decimal digits, when the (default) compiler option *NOEXTEND or the PROCESS statement option NOEXTEND is specified.
   - 31 decimal digits, when the arithmetic mode compiler option *EXTEND31 or PROCESS statement option EXTEND31 is specified.
   - 34 decimal digits, when the arithmetic mode compiler option *EXTEND31FULL or PROCESS statement option EXTEND31FULL is specified.
   - 63 decimal digits, when the arithmetic mode compiler option *EXTEND63 or PROCESS statement option EXTEND63 is specified.

   [End of IBM Extension]
2. If the value of i + d is an even number less than MAXLENGTH, the compiler converts it to an odd number by adding 1.
3. If the value of i + d exceeds 63, system message MCH1202 can result, even if the statement includes the SIZE ERROR phrase.

If you think an intermediate result field might exceed MAXLENGTH digits, you can use floating-point operands (COMP-1 and COMP-2) to avoid truncation.

Integer Functions

These functions always return an integer, and the outer-dmax will always be zero. For those functions whose arguments must be integer, the inner-dmax will also always be zero.

Table 46 on page 542 summarizes the precision of the function results:

Table 46. Precision of Integer Intrinsic Functions

Function	Inner Dmax	Outer Dmax	Function Result
DATE-OF-INTEGER	0	0	8-digit integer
DATE-TO-YYYYMMDD	0	0	9-digit integer
DAY-OF-INTEGER	0	0	7-digit integer
DAY-TO-YYYYDDD	0	0	9-digit integer

Table 46. Precision of Integer Intrinsic Functions (continued)

Function	Inner Dmax	Outer Dmax	Function Result
FIND-DURATION	N/A	0	9-digit integer
INTEGER-OF-DATE	0	0	7-digit integer
INTEGER-OF-DAY	0	0	7-digit integer
LENGTH	N/A	0	9- digit integer
ORD	N/A	0	3-digit integer
ORD-MAX	N/A	0	9-digit integer
ORD-MIN	N/A	0	9-digit integer
YEAR-TO-YYYY	0	0	9-digit integer

Table 47 on page 543 summarizes the precision of the function results:

Table 47. Precision of Integer Intrinsic Functions

Function	Inner Dmax	Outer Dmax	Function Result
DATE-OF-INTEGER	0	0	8-digit integer
DAY-OF-INTEGER	0	0	7-digit integer
FACTORIAL	0	0	fixed-point, 30-digit integer
INTEGER-OF-DATE	0	0	7-digit integer
INTEGER-OF-DAY	0	0	7-digit integer
LENGTH	N/A	0	9- digit integer
MOD	0	0	integer with as many digits as min(i1 i2)
ORD	N/A	0	3-digit integer
INTEGER		0	With a fixed-point argument, result will be fixed-point integer with one more integer digit than the argument. With a floating-point argument, result will be fixed-point, 30-digit integer.
INTEGER-PART		0	With a fixed-point argument, result will be fixed-point integer with the same number of integer digits as the argument. With a floating-point argument, result will be fixed-point, 30-digit integer.

Table 48 on page 543 summarizes the precision of the function results:

Table 48. Precision of Integer Intrinsic Functions

Function	Inner Dmax	Outer Dmax	Function Result
DATE-OF-INTEGER	0	0	8-digit integer
DAY-OF-INTEGER	0	0	7-digit integer
FACTORIAL	0	0	fixed-point, 30-digit integer
INTEGER-OF-DATE	0	0	7-digit integer

Table 48. Precision of Integer Intrinsic Functions (continued)

Function	Inner Dmax	Outer Dmax	Function Result
INTEGER-OF-DAY	0	0	7-digit integer
LENGTH	N/A	0	9- digit integer
MOD	0	0	integer with as many digits as min(i1 i2)
ORD	N/A	0	3-digit integer
ORD-MAX		0	9-digit integer
ORD-MIN		0	9-digit integer
INTEGER		0	With a fixed-point argument, result will be fixed-point integer with one more integer digit than the argument. With a floating-point argument, result will be fixed-point, 30-digit integer.
INTEGER-PART		0	With a fixed-point argument, result will be fixed-point integer with the same number of integer digits as the argument. With a floating-point argument, result will be fixed-point, 30-digit integer.

Mixed Functions

When the compiler handles a mixed function as fixed-point arithmetic, the result will be either integer or fixed-point with decimals (when any argument is floating-point, the function becomes a floating-point function and will follow floating-point rules). For MAX, MIN, RANGE, REM,and SUM, the outer-dmax is always equal to the inner-dmax. To determine the precision of the result returned for these functions, apply the rules for fixed-point arithmetic to each step in the algorithm used to calculate the function result.

MAX

1. Assign the first argument to your function result.
2. For each remaining argument:
   a. Compare the algebraic value of your function result with the argument.
   b. Assign the greater of the two to your function result.

MIN

1. Assign the first argument to your function result.
2. For each remaining argument:
   a. Compare the algebraic value of your function result with the argument.
   b. Assign the lesser of the two to your function result.

RANGE

1. Use the steps for MAX to select your maximum argument.
2. Use the steps for MIN to select your minimum argument.
3. Subtract the minimum argument from the maximum.
4. Assign the difference to your function result.

REM

1. Divide argument-1 by argument-2.
2. Remove all noninteger digits from the result of step 1.
3. Multiply the result of step 2 by argument-2.
4. Subtract the result of step 3 from argument-1.
5. Assign the difference to your function result.

SUM

1. Assign the value 0 to your function result.
2. For each argument:
   a. Add the argument to your function result.
   b. Assign the sum to your function result.

[IBM Extension] Floating-Point Data and Intermediate Results

Floating-point instructions are used to compute an arithmetic expression if any of the following conditions are true:

- A receiver or operand in the expression is COMP-1, COMP-2, external floating-point data, or a floating-point literal.
- An intrinsic numeric function is a floating-point function.
- The expression is an argument of a floating-point function.

If any operation in an arithmetic expression is computed in floating-point, the entire expression is computed as if all operands were converted to floating-point and the operations are evaluated using floating-point instructions.

If an expression is computed in floating-point, double-precision floating-point is used if any receiver or operand in the expression is not COMP-1, or if a multiplication or exponentiation operation appears in the expression. Whenever double-precision floating-point is used for one operation in an arithmetic expression, all operations in the expression are computed as if double-precision floating-point instructions were used.

Floating-point exponentiations are always evaluated using double-precision floating-point arithmetic.

The value of a negative number raised to a fractional power is undefined. For example, (-2) ** 3 is equal to -8, but (-2) ** (3.000001) is not defined. When an exponentiation is evaluated in floating-point and there is a possibility that the value of the exponentiation will be undefined (as in the example above), then the value of the exponent is evaluated at run time to determine if it is actually an integer.

The floating-point numeric functions will always return a double-precision floating-point value. For a list of the floating-point and fixed-point functions, see "Types of Numeric Functions" in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

Arithmetic expressions can appear in contexts other than arithmetic statements. For example, an arithmetic expression can be used with the IF statement. In such statements, the rules for intermediate results, floating-point, and double-precision floating-point apply, with the following changes:

- Abbreviated IF statements are handled as though the statements were not abbreviated.
- An explicit relation condition exists when a required relational operator is used to define the comparison between two operands (here referred to as comparands). In an explicit relation condition where one or both of the comparands is an arithmetic expression, the rules for intermediate results are determined by taking into consideration the attributes of both comparands. That is to say, dmax is defined to be the maximum number of decimal places defined for any operand of either comparand, except divisors and exponents. The rules for floating-point and double-precision floating-point apply if any operand in either comparand is COMP-1, COMP-2, external floating-point data, or a floating-point literal.

For example, in the statement:

```
IF operand-1 = expression-1 THEN . . .
```

where operand-1 is a data-name defined to be COMP-2, and expression-1 contains only fixed-point operands, the rules for floating-point arithmetic apply to expression-1 because it is being compared to a floating-point operand.

- When the comparison between an arithmetic expression and either a data item or another arithmetic expression is defined without using a relational operator, then no explicit relation condition is said to exist. In these cases, the comparison can be rewritten as one or more IF statements with an explicit operator. Each IF statement then follows the rules outlined above for an explicit relation condition. For example, in the statement:

```
EVALUATE expression-1
  WHEN expression-2 THRU expression-3
  WHEN expression-4
  .
  .
  .
END-EVALUATE
```

the equivalent IF statements are:

```
IF expression-1 >= expression-2 AND
   expression-1 <= expression-3
IF expression-1 = expression-4
   then for each IF statement, the comparands must be looked at
   to determine if all the arithmetic in that IF statement will
   be fixed-point or floating-point.
```

[End of IBM Extension]

Appendix C. 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	ƀ	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
. . .				

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
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
. . .				
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

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
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
. . .				
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

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
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
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	ƀ	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

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
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
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

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
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
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

Ordinal Number	Symbol	Meaning	Decimal Represen-tation	Hex Represen-tation
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	[a]	Split vertical bar	124	7C
126	}	Right brace	125	7D
127		Tilde	126	7E

Appendix D. ILE COBOL Function-Name and Context-Sensitive Word List

The following sections list all of the context-sensitive words and function-names in ILE COBOL.

Visual Key

The following key identifies the function-names and context-sensitive words in the ILE COBOL language:

Blank
An ILE COBOL function-name or context-sensitive word from Standard COBOL.

(1)
An ILE COBOL function-name or context-sensitive word that is an IBM extension to Standard COBOL.

(2)
A COBOL function-name from the 1985 (revised 1989) ANSI Standard that is not used by the ILE COBOL compiler.

Function-Names

Function-Name	Function-Name	Function-Name
ACOS	ADD-DURATION (1)	ANNUITY
ASIN	ATAN	CHAR
CONVERT-DATE-TIME (1)	COS	CURRENT-DATE
DATE-OF-INTEGER	DATE-TO-YYYYMMDD (1)	DAY-OF-INTEGER
DAY-TO-YYYYDDD (1)	DISPLAY-OF	EXTRACT-DATE-TIME (1)
FACTORIAL	FIND-DURATION (1)	INTEGER
INTEGER-OF-DATE	INTEGER-OF-DAY	INTEGER-PART
LENGTH	LOCALE-DATE (1)	LOCALE-TIME (1)
LOG	LOG10	LOWER-CASE
MAX	MEAN	MEDIAN
MIDRANGE	MIN	MOD
NATIONAL-OF	NUMVAL	NUMVAL-C
ORD	ORD-MAX	ORD-MIN
PRESENT-VALUE	RANDOM	RANGE
REM	REVERSE	SIN
SQRT	STANDARD-DEVIATION	SUBTRACT-DURATION (1)
SUM	TAN	TEST-DATE-TIME
TRIM (1)	TRIML (1)	TRIMR (1)
UPPER-CASE	UTF8STRING (1)	VARIANCE
WHEN-COMPILED	YEAR-TO-YYYY (1)	

[IBM Extension] Context-Sensitive Words

Context-Sensitive Word	Context
APPEND	`XML GENERATE FILE-STREAM APPEND data-1 FROM data-2`
DAYS	`MOVE FUNCTION ADD-DURATION(date-1 DAYS 90)` (Also can be used in SUBTRACT-DURATION, FIND-DURATION, and EXTRACT-DATE-TIME.)
DEFAULT	`SET LOCALE LC_ALL FROM DEFAULT`
HOURS	`MOVE FUNCTION ADD-DURATION(time-1 HOURS 90)` (Also can be used in SUBTRACT-DURATION, FIND-DURATION, and EXTRACT-DATE-TIME.)
LC_ALL	`SET LOCALE LC_ALL FROM DEFAULT`
LC_COLLATE	`SET LOCALE LC_COLLATE FROM DEFAULT`

Context-Sensitive Word	Context
LC_CURRENCY	`SET LOCALE LC_CURRENCY FROM DEFAULT`
LC_MESSAGES	`SET LOCALE LC_MESSAGES FROM DEFAULT`
LC_MONETARY	`SET LOCALE LC_MONETARY FROM DEFAULT`
LC_NUMERIC	`SET LOCALE LC_NUMERIC FROM DEFAULT`
LC_TIME	`SET LOCALE LC_TIME FROM DEFAULT`
LC_TYPE	`SET LOCALE LC_TYPE FROM DEFAULT`
MICROSECONDS	`MOVE FUNCTION ADD-DURATION(time-1 MICROSECONDS 30)` (Also can be used in SUBTRACT-DURATION, FIND-DURATION, and EXTRACT-DATE-TIME.)
MINUTES	`MOVE FUNCTION ADD-DURATION(time-1 MINUTES 35)` (Also can be used in SUBTRACT-DURATION, FIND-DURATION, and EXTRACT-DATE-TIME.)
MONTHS	`MOVE FUNCTION ADD-DURATION(date-1 MONTHS 12)` (Also can be used in SUBTRACT-DURATION, FIND-DURATION, and EXTRACT-DATE-TIME.)
OVERWRITE	`XML GENERATE FILE-STREAM OVERWRITE data-1 FROM data-2`
SECONDS	`MOVE FUNCTION ADD-DURATION(time-1 SECONDS 30)` (Also can be used in SUBTRACT-DURATION, FIND-DURATION, and EXTRACT-DATE-TIME.)
SYMBOL	`CURRENCY IS "EUR" PICTURE SYMBOL "$"`
TIMESTAMP	`05 date-1 FORMAT TIMESTAMP` (Also found in SPECIAL-NAMES paragraph, intrinsic functions TEST-DATE-TIME and CONVERT-DATE-TIME.)
YEARS	`MOVE FUNCTION ADD-DURATION(date-1 YEARS 2)` (Also can be used in SUBTRACT-DURATION, FIND-DURATION, and EXTRACT-DATE-TIME.)
YYYYDDD	`ACCEPT id-1 FROM DATE YYYYDDD`
YYYYMMDD	`ACCEPT id-1 FROM DATE YYYYMMDD`

[End of IBM Extension]

Appendix E. ILE COBOL Reserved Word List

The following sections list all of the reserved words in ILE COBOL.

Visual Key

The following key identifies the reserved words in the ILE COBOL language:

Blank
An ILE COBOL reserved word from Standard COBOL.

(1)
An ILE COBOL reserved word that is an IBM extension to the Standard COBOL.

(2)
A COBOL reserved word from Standard COBOL that is not used by the ILE COBOL compiler. These words should not be used if compatibility is important to an installation. If used, a diagnostic message will be issued.

(3)
A COBOL reserved word that is not in Standard COBOL and is not supported by the ILE COBOL compiler. If used, a diagnostic message will be issued.

Reserved Words

Reserved Word	Reserved Word
ACCEPT	ACCESS
ACQUIRE (1)	ADD
ADDRESS (1)	ADVANCING
AFTER	ALIAS (1)
ALL	ALPHABET
ALPHABETIC	ALPHABETIC-LOWER
ALPHABETIC-UPPER	ALPHANUMERIC
ALPHANUMERIC-EDITED	ALSO
ALTER	ALTERNATE
AND	ANY (2)
ARE	AREA
AREAS	ARITHMETIC (3)
ASCENDING	ASSIGN
AT	ATTRIBUTE (1)
AUTHOR	AUTO (1)
AUTO-SKIP (1)	AUTOMATIC (3)
BACKGROUND-COLOR (1)	BACKGROUND-COLOUR (1)
B-AND (3)	BEEP (1)
BEFORE	BELL (1)
B-EXOR (3)	BINARY
BIT (3)	BITS (3)

Reserved Word	Reserved Word
BLANK	B-LESS (3)
BLINK (1)	BLOCK
B-NOT (3)	BOOLEAN (3)
B-OR (3)	BOTTOM
BY	CALL
CANCEL	CD (2)
CF (2)	CH (2)
CHARACTER	CHARACTERS
CLASS	CLOCK-UNITS
CLOSE	COBOL (2)
CODE	CODE-SET
COL (1)	COLLATING
COLUMN	COMMA
COMMIT (1)	COMMITMENT (1)
COMMON	COMMUNICATION (2)
COMP	COMP-0 (3)
COMP-1 (1)	COMP-2 (1)
COMP-3 (1)	COMP-4 (1)
COMP-5 (1)	COMP-6 (3)
COMP-7 (3)	COMP-8 (3)
COMP-9 (3)	COMPUTATIONAL
COMPUTATIONAL-0 (3)	COMPUTATIONAL-1 (1)
COMPUTATIONAL-2 (1)	COMPUTATIONAL-3 (1)
COMPUTATIONAL-4 (1)	COMPUTATIONAL-5 (1)
COMPUTATIONAL-6 (3)	COMPUTATIONAL-7 (3)
COMPUTATIONAL-8 (3)	COMPUTATIONAL-9 (3)
COMPUTE	CONFIGURATION
CONNECT (3)	CONSOLE (1)
CONTAINED (3)	CONTAINS
CONTENT	CONTINUE
CONTROL	CONTROL-AREA (1)
CONTROLS	CONVERTING
COPY	CORR
CORRESPONDING	COUNT
CRT (1)	CRT-UNDER (1)
CURRENCY	CURRENT (3)
CURSOR (1)	DATA

Reserved Word	Reserved Word
DATE	DATE-COMPILED
DATE-WRITTEN	DAY
DAY-OF-WEEK	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 (1)
DBCS-EDITED (1)	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
DESCRIBED (1)	DESTINATION (2)
DETAIL (2)	DISABLE (2)
DISCONNECT (3)	DISPLAY
DISPLAY-1 (1)	DISPLAY-2 (3)
DISPLAY-3 (3)	DISPLAY-4 (3)
DISPLAY-5 (3)	DISPLAY-6 (3)
DISPLAY-7 (3)	DISPLAY-8 (3)
DISPLAY-9 (3)	DIVIDE
DIVISION	DOWN
DROP (1)	DUPLICATE (3)
DUPLICATES	DYNAMIC
EBCDIC (1)	EGI (2)
EJECT (1)	ELSE
EMI (2)	EMPTY (3)
EMPTY-CHECK (1)	ENABLE (2)
END	END-ACCEPT (1)
END-ADD	END-CALL
END-COMPUTE	END-DELETE
END-DISPLAY (1)	END-DIVIDE
END-EVALUATE	END-IF
END-INVOKE (1)	END-MULTIPLY

Reserved Word	Reserved Word
END-OF-PAGE	END-PERFORM
END-READ	END-RECEIVE (2)
END-RETURN	END-REWRITE
END-SEARCH	END-START
END-STRING	END-SUBTRACT
END-UNSTRING	END-WRITE
END-XML (1)	ENTER
ENTRY (1)	ENVIRONMENT
EOP	EQUAL
EQUALS (3)	ERASE (3)
ERROR	ESI (2)
EVALUATE	EVERY
EXCEEDS (3)	EXCEPTION
EXCLUSIVE (3)	EXIT
EXTEND	EXTERNAL
EXTERNALLY-DESCRIBED-KEY (1)	FALSE
FD	FETCH (3)
FILE	FILE-CONTROL
FILE-STREAM (1)	FILES (3)
FILLER	FINAL (2)
FIND (3)	FINISH (3)
FIRST	FOOTING
FOR	FOREGROUND-COLOR (1)
FOREGROUND-COLOUR (1)	FORMAT (1)
FREE (3)	FROM
FULL (1)	FUNCTION
GENERATE	GET (3)
GIVING	GLOBAL
GO	GOBACK (1)
GREATER	GROUP (2)
HEADING (2)	HIGHLIGHT (1)
HIGH-VALUE	HIGH-VALUES
I-O	I-O-CONTROL
ID (1)	IDENTIFICATION
IF	IN
INDEX	INDEXED
INDEX-1 (3)	INDEX-2 (3)

Reserved Word	Reserved Word
INDEX-3 (3)	INDEX-4 (3)
INDEX-5 (3)	INDEX-6 (3)
INDEX-7 (3)	INDEX-8 (3)
INDEX-9 (3)	INDIC (1)
INDICATE	INDICATOR (1)
INDICATORS (1)	INITIAL
INITIALIZE	INITIATE
INPUT	INPUT-OUTPUT
INSPECT	INSTALLATION
INTO	INVALID
INVOKE (1)	IS
JUST	JUSTIFIED
KANJI (1)	KEEP (3)
KEY	LABEL
LAST	LD (3)
LEADING	LEFT
LEFT-JUSTIFY (1)	LENGTH
LENGTH-CHECK (1)	LESS
LIBRARY (1)	LIKE (1)
LIMIT (2)	LIMITS (2)
LINAGE	LINAGE-COUNTER
LINE	LINE-COUNTER (2)
LINES	LINKAGE
LOCALE (1)	LOCALLY (3)
LOCAL-STORAGE (1)	LOCK
LOW-VALUE	LOW-VALUES
MEMBER (3)	MEMORY
MERGE	METACLASS (1)
MODE	MODIFIED (1)
MODIFY (3)	MODULES
MOVE	MULTIPLE
MULTIPLY	MESSAGE (2)
NATIONAL	NATIVE
NEGATIVE	NEXT
NO	NO-ECHO (1)
NONE (3)	NOT
NULL-KEY-MAP (1)	NULL-MAP (1)

Reserved Word	Reserved Word
NULL (1)	NULLS (1)
NUMBER	NUMERIC
NUMERIC-EDITED	OBJECT (1)
OBJECT-COMPUTER	OCCURS
OF	OFF
OMITTED	ON
ONLY (3)	OPEN
OPTIONAL	OR
ORDER	ORGANIZATION
OTHER	OUTPUT
OVERFLOW	OWNER (3)
PACKED-DECIMAL	PADDING
PAGE	PAGE-COUNTER (2)
PARSE (1)	PERFORM
PF (2)	PH (2)
PICTURE	PLUS (2)
PIC	POINTER
POSITION	POSITIVE
PREFIX (1)	PRESENT (3)
PRINTING	PRIOR (1)
PROCEDURE	PROCEDURE-POINTER (1)
PROCEDURES	PROCEED
PROCESS (1)	PROCESSING (1)
PROGRAM-ID	PROMPT (1)
PROGRAM	PROTECTED (3)
PURGE (2)	QUEUE (2)
QUOTE	QUOTES
RANDOM	RD (2)
READ	READY (3)
REALM (3)	RECEIVE (2)
RECURSIVE (1)	RECONNECT (3)
RECORD	RECORD-NAME (3)
RECORDS	REDEFINES
REEL	REFERENCE
REFERENCE-MONITOR (3)	REFERENCES
RELATION (3)	RELATIVE
RELEASE	REMAINDER

Reserved Word	Reserved Word
REMOVAL	RENAMES
REPEATED (3)	REPLACE
REPLACING	REPORT (2)
REPORTING (2)	REPORTS (2)
REPOSITORY (1)	REQUIRED (1)
RERUN	RESERVE
RESET	RETAINING (3)
RETRIEVAL (3)	RETURN
RETURNING (1)	RETURN-CODE (1)
REVERSED	REVERSE-VIDEO (1)
REWIND	REWRITE
RF (2)	RH (2)
RIGHT	RIGHT-JUSTIFY (1)
ROLLBACK (1)	ROLLING (1)
ROUNDED	RUN
SAME	SCREEN (1)
SD	SEARCH
SECTION	SECURE (1)
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 (1)	SOURCE (2)
SOURCE-COMPUTER	SPACE
SPACE-FILL (1)	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)

Reserved Word	Reserved Word
SUBFILE (1)	SUBSTITUTE (1)
SUBTRACT	SUM (2)
SUPPRESS	SYMBOLIC
SYNC	SYNCHRONIZED
SYSIN (1)	SYSOUT (1)
TABLE (2)	TALLYING
TAPE	TENANT (3)
TERMINAL	TERMINATE (2)
TEST	TEXT (2)
THAN	THEN
THROUGH	THRU
TIME	TIMES
TITLE (1)	TO
TOP	TRAILING
TRAILING-SIGN (1)	TRANSACTION (1)
TRUE	TYPE
TYPEDEF (1)	UNDERLINE (1)
UNEQUAL (3)	UNIT
UNSTRING	UNTIL
UP	UPDATE (1)
UPON	USAGE
USAGE-MODE (3)	USE
USING	VALID (3)
VALIDATE (3)	VALUE
VALUES	VARYING
VLR (1)	WAIT (3)
WHEN	WHEN-COMPILED (1)
WITH	WITHIN (3)
WORDS	WORKING-STORAGE
WRITE	XML (1)
XML-CODE (1)	XML-EVENT (1)
XML-NTEXT (1)	XML-TEXT (1)
ZERO	
ZEROES	ZERO-FILL (1)
ZEROS	<
<=	+
*	**

Reserved Word	Reserved Word
-	/
>	>=
=	

Appendix F. File Structure Support Summary and Status Key Values

File Structure Support Tables

Table 49 on page 563 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 50 on page 567 and Table 51 on page 568 contain status key values and their meanings.

Table 49. 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	Workstation	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	O

Table 49. File Structure Support (continued)

Device Type	Printer	Tape	Disk Seq	Disk Rel Seq	Disk Rel Random	Disk Rel Dynamic	Disk IDX Seq	Disk IDX Random	Disk IDX Dynamic	Workstation	Diskette	Format File
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	.	.
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	.
LINAGE	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	.

Table 49. File Structure Support (continued)

Device Type	Printer	Tape	Disk Seq	Disk Rel Seq	Disk Rel Random	Disk Rel Dynamic	Disk IDX Seq	Disk IDX Random	Disk IDX Dynamic	Workstation	Diskette	Format File
OUTPUT	R	O	O	O	O	O	O	O	O	.	O	O
I-O	.	.	O	O	O	O	O	O	O	R	.	.
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	.
NEXT	.	.	.	.	.	I	.	.	I	.	.	.
FIRST	.	.	.	.	.	.	.	.	D	.	.	.
LAST	.	.	.	.	.	.	.	.	D	.	.	.
PRIOR	.	.	.	.	.	.	.	.	D	.	.	.
INTO	.	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	.
NOT AT END	.	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	J	.	R
NULL-KEY-MAP	.	.	.	.	.	.	D	D	D	.	.	.
NULL-MAP	.	.	D	D	D	D	D	D	D	.	.	.
NEXT MODIFIED	.	.	.	.	.	.	.	.	.	B	.	.
SUBFILE	.	.	.	.	.	.	.	.	.	B	.	.
INDICATORS	.	.	.	.	.	.	.	.	.	J	.	.
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	.	.

Table 49. File Structure Support (continued)

Device Type	Printer	Tape	Disk Seq	Disk Rel Seq	Disk Rel Random	Disk Rel Dynamic	Disk IDX Seq	Disk IDX Random	Disk IDX Dynamic	Workstation	Diskette	Format File
ADVANCING	O	.	.	.	.	.	.	.	.	.	.	.
AT END-OF-PAGE	O	.	.	.	.	.	.	.	.	.	.	.
NOT AT END-OF-PAGE	O	.	.	.	.	.	.	.	.	.	.	.
FORMAT	.	.	D	.	.	.	D	D	D	R	.	R
NULL-KEY-MAP	.	.	.	.	.	.	D	D	D	.	.	.
NULL-MAP	.	.	D	D	D	D	D	D	D	.	.	.
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	.	.	.
NULL-KEY-MAP	.	.	.	.	.	.	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	.	.
NULL-KEY-MAP	.	.	.	.	.	.	D	D	D	.	.	.
NULL-MAP	.	.	D	D	D	D	D	D	D	.	.	.
INDICATORS	.	.	.	.	.	.	.	.	.	B	.	.
SUBFILE	.	.	.	.	.	.	.	.	.	S	.	.
TERMINAL	.	.	.	.	.	.	.	.	.	O	.	.
DELETE	.	.	.	O	O	O	O	O	O	.	.	.
NULL-KEY-MAP	.	.	.	.	.	.	D	D	D	.	.	.
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

Table 49. File Structure Support (continued)

Device Type	Printer	Tape	Disk Seq	Disk Rel Seq	Disk Rel Random	Disk Rel Dynamic	Disk IDX Seq	Disk IDX Random	Disk IDX Dynamic	Workstation	Diskette	Format File
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 *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

Table 50. File Status Keys and 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 OPTION(*CRTF) parameter description on the CRTCBLMOD command in the *ILE COBOL Programmer's Guide* for further information about dynamic file creation.)
0A	02 03	xx 09	Job being cancelled (controlled).
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.
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 the "Error and Exception Handling" section in the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide*.

Table 51. File Status Key Values

High Order Digit	Meaning	Low Order Digit	Meaning
0	Successful Completion	0	Nofurther 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 the accompanying notes under the READ statement.
		4	An attempt was made to read a record that is larger than the largest, or smaller than the smallest record allowed by the RECORD IS VARYING clause of the associated file-name.
		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. CPF4101, CPF4102, CPF4103, CPF4207, CPF9812.
		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.
		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).
		M	Last record written to a subfile. CPF5003
		P	The file has been opened successfully, but it contains null-capable fields and the ASSIGN clause does not specify ALWNULL and device-type DATABASE.
		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 51. File Status Key Values (continued)

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.

Table 51. File Status Key Values (continued)

High Order Digit	Meaning	Low Order Digit	Meaning
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: • The EXTEND or OUTPUT phrase was specified but the file would not support write operations. • The I-O phrase was specified but the file would not support the input and output operations permitted. • 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. The possible causes are: • The minimum record length specified by the program is less than the minimum record length required for the file. Level check error. CPF4131. • The file specifies the ALTERNATE RECORD KEY clause and one of the following errors was detected: 1. The field(s) in the database file that is to be used as an alternate record key is invalid. 2. The database file is a Distributed Data Management (DDM) file. 3. The database file allows it open data path to be shared. 4. The DUPLICATES clause specified for each key in the program does not match the duplicates attribute of the database file. This includes the primary key.

Table 51. File Status Key Values (continued)

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 that was already closed.
		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. An attempt was made to write or rewrite a record that is larger than the largest, or smaller than the smallest record allowed by the RECORD IS VARYING clause of the associated file-name.
		6	A sequential READ, READ NEXT or READ PRIOR 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 51. File Status Key Values (continued)

High Order Digit	Meaning	Low Order Digit	Meaning
9	Other errors	0	Other errors: • File not found • Member not found • The file specifies the ALTERNATE RECORD KEY clause and one of the following errors was detected: 1. A conflict was detected between an alternate record key open identifier and an existing one. 2. A permanent index cannot be found and the CRTARKIDX option was not specified. 3. The maximum number (156) of contiguous DDS fields used to form an alternate record key was exceeded. • 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 • CPF5001 through CPF5099 • CPF5101 through CPF5399 • CPF5501 through CPF5699 These exceptions are caught, and FILE STATUS is set to 90.
		1	Undefined or unauthorized access type CPF2207, CPF4104, CPF4236, CPF4238, CPF5057, CPF5109, CPF5134, CPF5279.
		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, 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.

Table 51. File Status Key Values (continued)

High Order Digit	Meaning	Low Order Digit	Meaning
9	Other errors	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).
		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 operating system 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.

Table 51. File Status Key Values (continued)

High Order Digit	Meaning	Low Order Digit	Meaning
9	Other errors	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, CPF5013, CPF5107, CPF5128, CPF5166, CPF5198, CPF5280, CPF5282, CPF5287, CPF5293, CPF5352, CPF5353, CPF5517, CPF5524, CPF5529, CPF5530, CPF5532, CPF5533, CPF5257.
		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.
		R	Referential integrity error. CPF502D, CPF502E, CPF503A.
		S	REWRITE or DELETE failed because last READ operation specified NO LOCK.
		T	Trigger program exception. CPF502B
		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.
		W	Check constraint exception. CPF502F.
		X	OPEN failed because the file type is not supported in a multithreaded job. Change the file type to DATABASE, PRINTER (spool file only), or a DDM file of type *IP and submit the program again. CPF4380.
		Y	OPEN failed because the auxiliary storage pool (ASP) device where the file is located is not available. CPF980B.

Attribute Data Formats

The layouts and values of the attribute data are system dependent. The following formats are for the ILE COBOL language.

For a complete list of device types and layouts, refer to the *Db2 for i* section of the *Database and File Systems* category in the **IBM i Information Center** at this Web site - http://www.ibm.com/systems/i/infocenter/.

```
 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
*          '3486BA' - 3486 MOD BA DISPLAY
*          '3487HA' - 3487 MOD HA DISPLAY
```

```
*          '3487HG' - 3487 MOD HG DISPLAY
*          '3487HW' - 3487 MOD HW DISPLAY
*          '3487HC' - 3487 MOD HC 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 - IDEOGRAPHIC
*          G - GRAPHIC DBCS
     02  KEYBOARD-TYPE              PIC X.
*          A - ALPHANUMERIC OR KATAKANA KEYBOARD
*          I - IDEOGRAPHIC 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)
*          2 - SYNCHRONIZATION LEVEL 2  (SYNLVL(*COMMIT))
```

```
     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  WORKSTATION-CONTROLLER     PIC X.
*          N - NOT ATTACHED
*          1 - ATTACHED TO CONTROLLER 1
*          2 - ATTACHED TO CONTROLLER 2
*          3 - ATTACHED TO CONTROLLER 3
     02  DISPLAY-IS-COLOR           PIC X.
*          Y - YES
*          N - NO
     02  DISPLAY-ALLOWS-GRID-LINES  PIC X.
*          N - NO
*          1 - YES
     02  LU6-CONVERSATION-STATE     PIC X.
*          '00'X - RESET
*          '01'X - SEND
*          '02'X - DEFER RECEIVED
*          '03'X - DEFER DEALLOCATE
*          '04'X - RECEIVE
*          '05'X - CONFIRM
*          '06'X - CONFIRM SEND
*          '07'X - CONFIRM DEALLOCATE
*          '08'X - COMMIT
*          '09'X - COMMIT SEND
*          '0A'X - COMMIT DEALLOCATE
*          '0B'X - DEALLOCATE
*          '0C'X - ROLLBACK REQUIRED
     02  LU6-CONVERSATION-CORRELATE PIC X(8).
     02  FILLER                     PIC X(31).
*          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
     02  TRANSACTION-PROGRAM-NAME       PIC X(64).
     02  LU6-PROTECTED-LUWID.
         03  LENGTH-OF-PROT-LUWID-FIELDS       PIC S9(4)    COMP-4.
         03  FILLER REDEFINES LENGTH-OF-PROT-LUWID-FIELDS.
             05  LENGTH-OF-PROT-LUWID-FIELD    PIC X.
             05  LENGTH-OF-PROT-LU-NAME        PIC X.
         03  NETWORK-QUAL-PROT-LU-NAME         PIC X(17).
         03  PROTECTED-INST-SEQ-NUMBERS.
             05  PROT-INSTANCE-NUMBER          PIC X(6).
             05  PROT-SEQUENCE-NUMBER          PIC S9(4)    COMP-4.
     02  LU6-UNPROTECTED-LUWID.
         03  LENGTH-OF-UNPROT-LUWID-FIELDS     PIC S9(4)    COMP-4.
         03  FILLER REDEFINES LENGTH-OF-UNPROT-LUWID-FIELDS.
             05  LENGTH-OF-UNPROT-LUWID-FIELD  PIC X.
             05  LENGTH-OF-UNPROT-LU-NAME      PIC X.
         03  NETWORK-QUAL-UNPROT-LU-NAME       PIC X(17).
         03  UNPROTECTED-INST-SEQ-NUMBERS.
             05  UNPROT-INSTANCE-NUMBER        PIC X(6).
             05  UNPROT-SEQUENCE-NUMBER        PIC S9(4)    COMP-4.
```

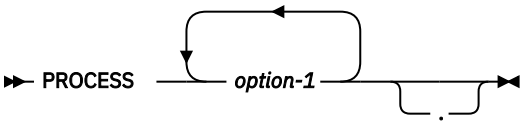
Appendix G. PROCESS Statement

The PROCESS statement is an optional part of the COBOL source program. It lets you specify options that you would normally spencify at compilation time.

Options specified in the PROCESS statement **override** the corresponding options specified in the CRTCBLMOD or CRTBNDCBL CL command.

The format of the PROCESS statement is as follows:

PROCESS Statement - Format



Corresponding Create Command Options

The following tables indicate the allowable PROCESS statement options and the equivalent CRTCBLMOD and CRTBNDCBL command parameters and options. The defaults are underlined.

Descriptions of the PROCESS statement options correspond to the parameter and option descriptions for the CRTCBLMOD and CRTBNDCBL parameters. See the *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide* for more information.

PROCESS Statement Options	CRTCBLMOD/CRTBNDCBL
	OUTPUT Parameter Options
OUTPUT NOOUTPUT	***PRINT** *NONE

PROCESS Statement Option	CRTCBLMOD/CRTBNDCBL
	GENLVL Parameter Option
GENLVL(nn)	nn

PROCESS Statement Options	CRTCBLMOD/CRTBNDCBL
	OPTION Parameter Options
SOURCE **SRC** NOSOURCE NOSRC	***SOURCE** ***SRC** *NOSOURCE *NOSRC
NOXREF XREF	***NOXREF** *XREF
GEN NOGEN	***GEN** *NOGEN
NOSEQUENCE SEQUENCE	***NOSEQUENCE** *SEQUENCE
NOVBSUM VBSUM	***NOVBSUM** *VBSUM
NONUMBER NUMBER LINENUMBER	***NONUMBER** *NUMBER *LINENUMBER
NOMAP MAP	***NOMAP** *MAP
NOOPTIONS OPTIONS	***NOOPTIONS** *OPTIONS
QUOTE APOST	***QUOTE** *APOST
NOSECLVL SECLVL	***NOSECLVL** *SECLVL
PRTCORR NOPRTCORR	***PRTCORR** *NOPRTCORR

PROCESS Statement Options	CRTCBLMOD/CRTBNDCBL
	OPTION Parameter Options
MONOPRC NOMONOPRC	***MONOPRC** *NOMONOPRC
RANGE NORANGE	***RANGE** *NORANGE
NOUNREF UNREF	***NOUNREF** *UNREF
NOSYNC SYNC	***NOSYNC** *SYNC
NOCRTF CRTF	***NOCRTF** *CRTF
NODUPKEYCHK DUPKEYCHK	***NODUPKEYCHK** *DUPKEYCHK
NOINZDLT INZDLT	***NOINZDLT** *INZDLT
NOBLK BLK	***NOBLK** *BLK
STDINZ NOSTDINZ STDINZHEX00	***STDINZ** *NOSTDINZ *STDINZHEX00
NODDSFILLER DDSFILLER	***NODDSFILLER** *DDSFILLER
Not applicable	***NOIMBEDERR** *IMBEDERR
STDTRUNC NOSTDTRUNC	***STDTRUNC** *NOSTDTRUNC
CHGPOSSGN NOCHGPOSSGN	***CHGPOSSGN** *NOCHGPOSSGN
Not applicable	***NOEVENTF** *EVENTF
MONOPIC NOMONOPIC	***MONOPIC** *NOMONOPIC

PROCESS Statement Options	CRTCBLMOD/CRTBNDCBL
	OPTION Parameter Options
NOCRTARKIDX CRTARKIDX	***NOCRTARKIDX** *CRTARKIDX

PROCESS Statement Options	CRTCBLMOD/CRTBNDCBL
	CVTOPT Parameter Options
NOVARCHAR VARCHAR	***NOVARCHAR** *VARCHAR
NODATETIME DATETIME	***NODATETIME** *DATETIME
NOCVTPICXGRAPHIC CVTPICXGRAPHIC CVTPICGGRAPHIC NOCVTPICGGRAPHIC	***NOPICXGRAPHIC** *PICXGRAPHIC *PICGGRAPHIC *NOPICGGRAPHIC
NOCVTPICNGRAPHIC CVTPICNGRAPHIC	***NOPICNGRAPHIC** *PICNGRAPHIC
NOFLOAT FLOAT	***NOFLOAT** *FLOAT
NODATE DATE	***NODATE** *DATE
NOTIME TIME	***NOTIME** *TIME
NOTIMESTAMP TIMESTAMP	***NOTIMESTAMP** *TIMESTAMP
NOCVTTODATE CVTTODATE	***NOCVTTODATE** *CVTTODATE

PROCESS Statement Options	CRTCBLMOD/CRTBNDCBL
	OPTIMIZE Parameter Options
NOOPTIMIZE BASICOPT FULLOPT NEVEROPTIMIZE	***NONE** *BASIC *FULL *NEVER

PROCESS Statement Options	CRTCBLMOD/CRTBNDCBL
	FLAGSTD Parameter Options
NOFIPS MINIMUM INTERMEDIATE HIGH	***NOFIPS** *MINIMUM *INTERMEDIATE *HIGH
NOOBSOLETE OBSOLETE	***NOOBSOLETE** *OBSOLETE

PROCESS Statement Options EXTDSPOPT(*a b c*)	CRTCBLMOD/CRTBNDCBL
	EXTDSPOPT Parameter Options
DFRWRT NODFRWRT	***DFRWRT** *NODFRWRT
UNDSPCHR NOUNDSPCHR	***UNDSPCHR** *NOUNDSPCHR
ACCUPDALL ACCUPDNE	***ACCUPDALL** *ACCUPDNE

PROCESS Statement Option	CRTCBLMOD/CRTBNDCBL
	FLAG Parameter Option
FLAG(nn)	nn

PROCESS Statement Options	CRTCBLMOD/CRTBNDCBL
	LINKLIT Parameter Options
LINKPGM LINKPRC	***PGM** *PRC

PROCESS Statement Options SRTSEQ(*a*)	CRTCBLMOD/CRTBNDCBL
	SRTSEQ Parameter Options
HEX JOB JOBRUN LANGIDUNQ LANGIDSHR "LIBL/sort-seq-table-name" "CURLIB/sort-seq-table-name" "library-name/sort-seq-table-name" "sort-seq-table-name"	***HEX** *JOB *JOBRUN *LANGIDUNQ *LANGIDSHR *LIBL/sort-seq-table-name *CURLIB/sort-seq-table-name library-name/sort-seq-table-name sort-seq-table-name

PROCESS Statement Options LANGID(*a*)	CRTCBLMOD/CRTBNDCBL
	LANGID Parameter Options
JOBRUN JOB "language-identifier-name"	***JOBRUN** *JOB language-identifier-name

PROCESS Statement Options ENBPFRCOL(*a*)	CRTCBLMOD/CRTBNDCBL
	ENBPFRCOL Parameter Options
PEP ENTRYEXIT FULL	***PEP** *ENTRYEXIT *FULL

PROCESS Statement Options PRFDTA(*a*)	CRTCBLMOD/CRTBNDCBL
	PRFDTA Parameter Options
NOCOL COL	***NOCOL** *COL

PROCESS Statement Options CCSID(*a b c d*)	CRTCBLMOD/CRTBNDCBL
	CCSID Parameter Options
a = Locale single-byte data CCSID	
JOBRUN JOB HEX *coded-character-set-identifier*	***JOBRUN** *JOB *HEX *coded-character-set-identifier*
b = Non-locale single-byte data CCSID	
CCSID (uses CCSID specified for “a” above) JOBRUN JOB HEX *coded-character-set-identifier*	Not applicable
c = Non-locale double-byte data CCSID	
CCSID (uses CCSID specified for “a” above) JOBRUN JOB HEX *coded-character-set-identifier*	Not applicable
d = XML GENERATE single-byte or unicode data output CCSID	

PROCESS Statement Options CCSID(*a b c d*)	CRTCBLMOD/CRTBNDCBL
	CCSID Parameter Options
JOBRUN CCSID (uses CCSID specified for “a” above) JOB HEX *coded-character-set-identifier*	Not applicable

PROCESS Statement Option NTLCCSID(a)	CRTCBLMOD/CRTBNDCBL
	NTLCCSID Parameter Options
13488 coded-character-set-identifier	**13488** coded-character-set-identifier

PROCESS Statement Options DATTIM(*a b*)	CRTCBLMOD/CRTBNDCBL
4-digit base century (default 1900) *2-digit base year* (default 40)	Not applicable

PROCESS Statement Options THREAD(*a*)	CRTCBLMOD/CRTBNDCBL
NOTHREAD SERIALIZE	Not applicable

PROCESS Statement Options ARITHMETIC(*a*)	CRTCBLMOD/CRTBNDCBL
	ARITHMETIC Parameter Options
NOEXTEND EXTEND31 EXTEND31FULL EXTEND63	***NOEXTEND** *EXTEND31 *EXTEND31FULL *EXTEND63

PROCESS Statement Option	CRTCBLMOD/CRTBNDCBL
NOGRAPHIC GRAPHIC	Not applicable

PROCESS Statement Option	CRTCBLMOD/CRTBNDCBL
NONATIONAL NATIONAL NATIONALPICNLIT	Not applicable

PROCESS Statement Option	CRTCBLMOD/CRTBNDCBL
NOLSPTRALIGN LSPTRALIGN	Not applicable

PROCESS Statement Option	CRTCBLMOD/CRTBNDCBL
NOCOMPASBIN COMPASBIN	Not applicable

PROCESS Statement Option	CRTCBLMOD/CRTBNDCBL
	DBGVIEW Parameter Options
NOCOMPRESSDBG COMPRESSDBG	***NOCOMPRESSDBG** *COMPRESSDBG

PROCESS Statement Option OPTVALUE(*a*)	CRTCBLMOD/CRTBNDCBL
NOOPT OPT	Not applicable

PROCESS Statement Option	CRTCBLMOD/CRTBNDCBL
NOADJFILLER ADJFILLER	Not applicable

PROCESS Statement Option NTLPADCHAR(a b c)	CRTCBLMOD/CRTBNDCBL
	NTLPADCHAR Parameter Options
a = padding character for moving single-byte to national	
NX"0020" a national hexadecimal literal representing one national character	**NX"0020"** a national character
b = padding character for moving double-byte to national	
NX"3000" a national hexadecimal literal representing one national character	**NX"3000"** a national character
c = padding character for moving national to national	
NX"3000" a national hexadecimal literal representing one national character	**NX"3000"** a national character

PROCESS Statement Option LICOPT(a)	CRTCBLMOD/CRTBNDCBL
	LICOPT Parameter Option
licensed-internal-code-option-string	licensed-internal-code-option-string

PROCESS Statement Option PGMINFO(a b)	CRTCBLMOD/CRTBNDCBL
	PGMINFO Parameter Options
a = program interface information to be generated	

PROCESS Statement Option PGMINFO(a b)	CRTCBLMOD/CRTBNDCBL PGMINFO Parameter Options
NOPGMINFO PCML	***NO** *PCML
b = location for the generated program information	
MODULE	***STMF** *MODULE *ALL

PROCESS Statement Options STGMDL(*a*)	CRTCBLMOD STGMDL Parameter Options
INHERIT SNGLVL TERASPACE	***INHERIT** *SNGLVL *TERASPACE

PROCESS Statement Options STGMDL(*a*)	CRTBNDCBL STGMDL Parameter Options
SNGLVL INHERIT TERASPACE	***SNGLVL** *INHERIT *TERASPACE

PROCESS Statement Options ACTGRP(*a*)	CRTBNDCBL ACTGRP Parameter Options
STGMDL NEW CALLER 'activation-group-name'	***STGMDL** *NEW *CALLER activation-group-name

Appendix H. Complex OCCURS DEPENDING ON

Complex OCCURS DEPENDING ON (ODO) is supported as an extension to the COBOL 85 Standard.

The basic forms of complex ODO permitted by the compiler are as follows:

- Variably located item or group: A data item described by an OCCURS clause with the DEPENDING ON option is followed by a nonsubordinate data item or group.
- Variably located table: A data item described by an OCCURS clause with the DEPENDING ON option is followed by a nonsubordinate data item described by an OCCURS clause.
- Table with variable-length elements: A data item described by an OCCURS clause contains a subordinate data item described by an OCCURS clause with the DEPENDING ON option.
- Index name for a table with variable-length elements.
- Element of a table with variable-length elements.

Complex ODO can help you save disk space, but it can be tricky to use and can make maintaining your code more difficult.

The following example illustrates the possible types of occurrence of complex ODO:

```
01  FIELD-A.
    02 COUNTER-1                          PIC S99.
    02 COUNTER-2                          PIC S99.
    02 TABLE-1.
       03 RECORD-1 OCCURS 1 TO 5 TIMES
                   DEPENDING ON COUNTER-1    PIC X(3).
    02 EMPLOYEE-NUMBER                    PIC X(5).  1
    02 TABLE-2 OCCURS 5 TIMES                        2 3
              INDEXED BY INDX.                      4
       03 TABLE-ITEM                      PIC 99.    5
       03 RECORD-2 OCCURS 1 TO 3 TIMES
                   DEPENDING ON COUNTER-2.
          04 DATA-NUM                     PIC S99.
```

In the example, COUNTER-1 is called an ODO object because it is the object of the DEPENDING ON clause of RECORD-1. RECORD-1 is called an ODO subject. Similarly, COUNTER-2 is the ODO object of the corresponding ODO subject, RECORD-2.

The types of complex ODO occurrences shown in the example above are as follows:

1
A variably located item: EMPLOYEE-NUMBER is a data item following, but not subordinate to, a variable-length table in the same level-01 record.

2
A variably located table: TABLE-2 is a table following, but not subordinate to, a variable-length table in the same level-01 record.

3
A table with variable-length elements: TABLE-2 is a table containing a subordinate data item, RECORD-2, whose number of occurrences varies depending on the content of its ODO object.

4
An index name, INDX, for a table with variable-length elements.

5
An element, TABLE-ITEM, of a table with variable-length elements.

The length of the variable portion of each record is the product of its ODO object and the length of its ODO subject. For example, whenever a reference is made to one of the complex ODO items shown above, the actual length, if used, is computed as follows:

- The length of TABLE-1 is calculated by multiplying the contents of COUNTER-1 (the number of occurrences of RECORD-1) by 3 (the length of RECORD-1).
- The length of TABLE-2 is calculated by multiplying the contents of COUNTER-2 (the number of occurrences of RECORD-2) by 2 (the length of RECORD-2), and adding the length of TABLE-ITEM.
- The length of FIELD-A is calculated by adding the lengths of COUNTER-1, COUNTER-2, TABLE-1, EMPLOYEE-NUMBER, and TABLE-2 times 5.

You must set every ODO object in a group before you reference any complex ODO item in the group. For example, before you refer to EMPLOYEE-NUMBER in the code above, you must set COUNTER-1 and COUNTER-2 even though EMPLOYEE-NUMBER does not directly depend on either ODO object for its value.

Effects of a Change in ODO Value

If a data item described by an OCCURS clause with the DEPENDING ON option is followed in the same group by one or more nonsubordinate data items (a form of complex ODO), any change in value of the ODO object affects subsequent references to complex ODO items in the record:

- The size of any group containing the relevant ODO clause reflects the new value of the ODO object.
- A MOVE to a group containing the ODO subject is made based on the new value of the ODO object.

- The location of any nonsubordinate items following the item described with the ODO clause is affected by the new value of the ODO object. (To preserve the contents of the nonsubordinate items, move them to a work area before the value of the ODO object changes; then move them back.)

The value of an ODO object can change when you move data to the ODO object or to the group in which it is contained. The value can also change if the ODO object is contained in a record that is the target of a READ statement.

Preventing Errors when Changing the ODO Object Value

Be careful if you reference a complex-ODO index name (an index name for a table with variable-length elements) after having changed the value of the ODO object for a subordinate data item in the table. When you change the value of an ODO object, the byte offset in an associated complex-ODO index is no longer valid because the table length has changed. Unless you take precautions, you will obtain unexpected results if you then code a reference to the index name such as in the following:

- A reference to an element of the table
- A SET statement of the form SET *integer-data-item* TO *index-name* (format-1)
- A SET statement of the form SET *index-name* UP|DOWN BY *integer* (format-2)

To avoid this type of error, do the following:

1. Save the index item in an integer data item. (Doing so causes an implicit conversion: the integer item receives the table element occurrence number corresponding to the offset in the index.)
2. Change the value of the ODO object.
3. Immediately restore the index item from the integer data item. (Doing so causes an implicit conversion: the index item receives the offset corresponding to the table element occurrence number in the integer item. The offset is computed according to the table length then in effect.)

The following code shows how to save and restore the index name shown in “Appendix H. Complex OCCURS DEPENDING ON” on page 585 when the ODO object COUNTER-2 changes.

```
 77  INTEGER-DATA-ITEM-1      PIC 99.
 ...
     SET INDX TO 5.
*         INDX is valid at this point.
     SET INTEGER-DATA-ITEM-1 TO INDX.
*         INTEGER-DATA-ITEM-1 now has the
*         occurrence number corresponding to INDX.
     MOVE NEW-VALUE TO COUNTER-2.
*         INDX is not valid at this point.
     SET INDX TO INTEGER-DATA-ITEM-1.
*         INDX is now valid, containing the offset
*         corresponding to INTEGER-DATA-ITEM-1, and
*         can be used with the expected results.
```

Preventing Overlay When Adding Elements to a Variable Table

Be careful if you increase the number of elements in a variable-occurrence table that is followed by one or more nonsubordinate data items in the same group. When you increment the value of the ODO object and add an element to a table, you can inadvertently overlay the variably located data items that follow the table.

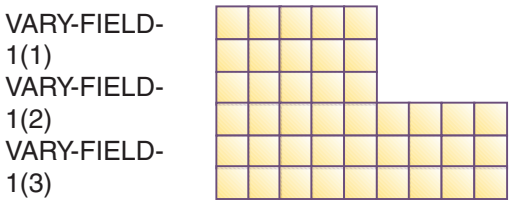
To avoid this type of error, do the following:

1. Save the variably located data items that follow the table in another data area.
2. Increment the value of the ODO object.
3. Move data into the new table element (if needed).
4. Restore the variably located data items from the data area where you saved them.

In the following example, suppose you want to add an element to the table VARY-FIELD-1, whose number of elements depends on the ODO object CONTROL-1. VARY-FIELD-1 is followed by the nonsubordinate variably located data item GROUP-ITEM-1, whose elements could potentially be overlaid.

```
WORKING-STORAGE SECTION.
01  VARIABLE-REC.
    05  FIELD-1                              PIC X(10).
    05  CONTROL-1                            PIC S99.
    05  CONTROL-2                            PIC S99.
    05  VARY-FIELD-1 OCCURS 1 TO 10 TIMES
          DEPENDING ON CONTROL-1              PIC X(5).
    05  GROUP-ITEM-1.
        10  VARY-FIELD-2
              OCCURS 1 TO 10 TIMES
              DEPENDING ON CONTROL-2          PIC X(9).
01  STORE-VARY-FIELD-2.
    05  GROUP-ITEM-2.
        10  VARY-FLD-2
              OCCURS 1 TO 10 TIMES
              DEPENDING ON CONTROL-2          PIC X(9).
```

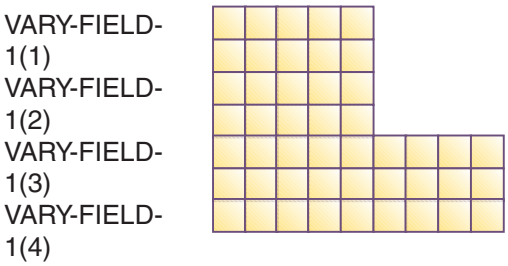
Each element of VARY-FIELD-1 has 5 bytes, and each element of VARY-FIELD-2 has 9 bytes. If CONTROL-1 and CONTROL-2 both contain the value 3, you can picture storage for VARY-FIELD-1 and VARY-FIELD-2 as follows:



To add a fourth element to VARY-FIELD-1, code as follows to prevent overlaying the first 5 bytes of VARY-FIELD-2. (GROUP-ITEM-2 serves as temporary storage for the variably located GROUP-ITEM-1.)

```
MOVE GROUP-ITEM-1 TO GROUP-ITEM-2.
ADD 1 TO CONTROL-1.
MOVE five-byte-field TO
  VARY-FIELD-1 (CONTROL-1).
MOVE GROUP-ITEM-2 TO GROUP-ITEM-1.
```

You can picture the updated storage for VARY-FIELD-1 and VARY-FIELD-2 as follows:



Note that the fourth element of VARY-FIELD-1 did not overlay the first element of VARY-FIELD-2.

Appendix I. ACCEPT/DISPLAY and COBOL/2 Considerations

The ILE COBOL 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 ILE COBOL 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 ILE COBOL extended ACCEPT statement but the COBOL/2 compiler generates a severe error message.
- BELL may be specified with a group item on a ILE COBOL extended ACCEPT statement but the COBOL/2 compiler generates a severe error message.
- The ILE COBOL 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/2 compiler justifies the signed numeric data (displayed and accepted) to the left, and the ILE COBOL 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 ILE COBOL 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 ILE COBOL 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 ILE COBOL extended ACCEPT or DISPLAY statement. (Error message LNC1263 is issued at compilation time, and LNR7054 at run time.)
- 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 (LNC0265) 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 EXTDSPOPT(*NODFRWRT) parameter in the CRTCBLMOD or CRTBNDCBL command, the ILE COBOL compiler buffers all extended DISPLAY statements until the next ACCEPT statement is encountered.
- Under the *NOUNDSPCHR compiler option, values below hexadecimal 20 cause undesirable results in extended ACCEPT and extended DISPLAY operations. To overcome this hardware limitation, use the (default) *UNDSPCHR option.
- The ILE COBOL 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 ILE COBOL compiler is 6 bytes.

Chapter 10. Bibliography

For additional information about topics related to ILE COBOL programming on the IBM i system, refer to the following IBM i publications:

- *Communications Management, SC41-5406-02*, provides information about using the Application Development ToolSet programming development manager (PDM) to work with lists of libraries, objects, members, and user-defined options to easily do such operations as copy, delete, and rename. Contains activities and reference material to help the user learn PDM. The most commonly used operations and function keys are explained in detail using examples.
- *ADTS for AS/400: Source Entry Utility, SC09-2605-00*, provides information about using the Application Development ToolSet source entry utility (SEU) to create and edit source members. The manual explains how to start and end an SEU session and how to use the many features of this full-screen text editor. The manual contains examples to help both new and experienced users accomplish various editing tasks, from the simplest line commands to using pre-defined prompts for high-level languages and data formats.
- *Application Display Programming, SC41-5715-02*, provides information about:
  - Using DDS to create and maintain displays for applications;
  - Creating and working with display files on the system;
  - Creating online help information;
  - Using UIM to define panels and dialogs for an application;
  - Using panel groups, records, or documents
- *Recovering your system, SC41-5304-10*, provides information about setting up and managing the following:
  - Journaling, access path protection, and commitment control
  - User auxiliary storage pools (ASPs)
  - Disk protection (device parity, mirrored, and checksum)

  Provides performance information about backup media and save/restore operations. Also includes advanced backup and recovery topics, such as using save-while-active support, saving and restoring to a different release, and programming tips and techniques.
- *CICS for iSeries Application Programming Guide, SC41-5454-02*, provides information on application programming for CICS® for IBM i. It includes guidance and reference information on the CICS application programming interface and system programming interface commands, and gives general information about developing new applications and migrating existing applications from other CICS platforms.
- *CL Programming, SC41-5721-06*, provides a wide-ranging discussion of IBM i programming topics including a general discussion on objects and libraries, CL programming, controlling flow and communicating between programs, working with objects in CL programs, and creating CL programs. Other topics include predefined and impromptu messages and message handling, defining and creating user-defined commands and menus, application testing, including debug mode, breakpoints, traces, and display functions.
- *Communications Management, SC41-5406-02*, provides information about work management in a communications environment, communications status, tracing and diagnosing communications problems, error handling and recovery, performance, and specific line speed and subsystem storage information.
- *Experience RPG IV Tutorial, GK2T-9882-00*, is an interactive self-study program explaining the differences between RPG III and RPG IV and how to work within the new ILE environment. An accompanying workbook provides additional exercises and doubles as a reference upon completion of the tutorial. ILE RPG code examples are shipped with the tutorial and run directly on the system.

- *GDDM Programming Guide, SC41-0536-00*, provides information about using IBM i graphical data display manager (GDDM) to write graphics application programs. Includes many example programs and information to help users understand how the product fits into data processing systems.
- *GDDM Reference, SC41-3718-00*, provides information about using IBM i graphical data display manager (GDDM) to write graphics application programs. This manual provides detailed descriptions of all graphics routines available in GDDM. Also provides information about high-level language interfaces to GDDM.
- *ICF Programming, SC41-5442-00*, provides information needed to write application programs that use IBM i communications and the IBM i intersystem communications function (IBM i-ICF). Also contains information on data description specifications (DDS) keywords, system-supplied formats, return codes, file transfer support, and program examples.
- *IDDU Use, SC41-5704-00*, describes how to use the IBM i interactive data definition utility (IDDU) to describe data dictionaries, files, and records to the system. Includes:
  - An introduction to computer file and data definition concepts
  - An introduction to the use of IDDU to describe the data used in queries and documents
  - Representative tasks related to creating, maintaining, and using data dictionaries, files, record formats, and fields
  - Advanced information about using IDDU to work with files created on other systems and information about error recovery and problem prevention.
- *IBM Rational Development Studio for i: ILE C/C++ Programmer's Guide, SC09-2712-07*, provides information on how to develop applications using the ILE C language. It includes information about creating, running and debugging programs. It also includes programming considerations for interlanguage program and procedure calls, locales, handling exceptions, database, externally described and device files. Some performance tips are also described. An appendix includes information on migrating source code from EPM C/400 or System C/400 to ILE C.
- *IBM Rational Development Studio for i: ILE C/C++ Language Reference, SC09-7852-02*, describes the syntax, semantics, and IBM implementation of the C and C++ programming languages.
- *IBM Rational Development Studio for i: ILE COBOL Programmer's Guide, SC09-2540-07*, provides information about how to write, compile, bind, run, debug, and maintain ILE COBOL programs on the IBM i system. It provides programming information on how to call other ILE COBOL and non-ILE COBOL programs, share data with other programs, use pointers, and handle exceptions. It also describes how to perform input/output operations on externally attached devices, database files, display files, and ICF files.
- *ILE Concepts, SC41-5606-09*, explains concepts and terminology pertaining to the Integrated Language Environment (ILE) architecture of the IBM i licensed program. Topics covered include creating modules, binding, running programs, debugging programs, and handling exceptions.
- *IBM Rational Development Studio for i: ILE RPG Programmer's Guide, SC09-2507-08*, provides information about the ILE RPG programming language, which is an implementation of the RPG IV language in the Integrated Language Environment (ILE) on the IBM i system. It includes information on creating and running programs, with considerations for procedure calls and interlanguage programming. The guide also covers debugging and exception handling and explains how to use IBM i files and devices in RPG programs. Appendixes include information on migration to RPG IV and sample compiler listings. It is intended for people with a basic understanding of data processing concepts and of the RPG language.
- *IBM Rational Development Studio for i: ILE RPG Reference, SC09-2508-08*, provides information about the ILE RPG programming language. This manual describes, position by position and keyword by keyword, the valid entries for all RPG IV specifications, and provides a detailed description of all the operation codes and built-in functions. This manual also contains information on the RPG logic cycle, arrays and tables, editing functions, and indicators.
- *Local Device Configuration, SC41-5121-00*, provides information about configuring local devices on the IBM i system. This includes information on how to configure the following:
  - Local work station controllers (including twinaxial controllers)

  - Tape controllers
  - Locally attached devices (including twinaxial devices)
- *Printer Device Programming, SC41-5713-06*, provides information to help you understand and control printing. Provides specific information on printing elements and concepts of the IBM i system, printer file and print spooling support for printing operations, and printer connectivity. Includes considerations for using personal computers, other printing functions such as Business Graphics Utility (BGU), advanced function printing (AFP™), and examples of working with the IBM i system printing elements such as how to move spooled output files from one output queue to a different output queue. Also includes an appendix of control language (CL) commands used to manage printing workload. Fonts available for use with the IBM i system are also provided. Font substitution tables provide a cross-reference of substituted fonts if attached printers do not support application-specified fonts.
- *ADTS for AS/400: Screen Design Aid, SC09-2604-00*, provides information about using the Application Development ToolSet screen design aid (SDA) to design, create, and maintain displays, menus, and online help information. The manual contains examples and information to help learn how to use SDA on the IBM i system and in the System/38 environment of the IBM i system.
- *Security reference, SC41-5302-11*, tells how system security support can be used to protect the system and the data from being used by people who do not have the proper authorization, protect the data from intentional or unintentional damage or destruction, keep security information up-to-date, and set up security on the system.
- *Installing, upgrading, or deleting IBM i and related software, SC41-5120-11*, provides step-by-step procedures for initial installation, installing licensed programs, program temporary fixes (PTFs), and secondary languages from IBM. This manual is also for users who already have an IBM i system with an installed release and want to install a new release.

For information about Systems Application Architecture (SAA) Common Programming Interface (CPI) COBOL, refer to the following publication:

- *Systems Application Architecture Common Programming Interface COBOL Reference*, SC26-4354.

Chapter 11. Acknowledgments

IBM acknowledges the use of the following research product in the ILE COBOL compiler:

S/SL

©Copyright 1981 by the University of Toronto

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information 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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_.

Programming interface information

This ILE COBOL Language Reference publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks of Oracle, Inc. in the United States, other countries, or both.

Other product and service names might be trademarks of IBM or other companies.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

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

Y

Z

SC09-2539-08