Elastic COBOL Language Reference Manual

COBOL-85 Standard ANSI X3.23B

REVISION: OCTOBER 2015

Preface

Trademarks

- IBM is a registered trademark of International Business Machines Inc.
- Oracle and Java are registered trademarks of Oracle and/or its affiliates.
- UNIX is a registered trademark licensed exclusively to X/Open Company Limited.
- Linux is a registered trademark of Linus Torvalds.
- Windows is a registered trademark of Microsoft Corporation.
- Eclipse is a trademark of the Eclipse Foundation Inc.
- Other names may be trademarks of their respective owners.

COBOL Language Reference

This describes the base COBOL language supported by this system: this COBOL language is based on the ANSI COBOL standards X3.23-1985, X3.23a-1989 and X3.23b-1993, and is supported by a number of COBOL systems. In addition, support has been added for some of the features from ISO/IEC 1989:2002, Programming language COBOL.

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

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

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

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

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

Contents

Table of Figures

1. Scope

Programming language COBOL

Information technology - Programming languages, their environments and system software interfaces.

This documentation specifies the syntax and meaning of programs written in Elastic COBOL. Its purpose is to promote a high degree of machine independence in COBOL programs to permit their use on a variety of data processing systems. Machine independence is achieved by executing COBOL in the Java Virtual Machine (JVM).

This documentation specifies:

- The form of a program written in Elastic COBOL.
- The effect of compiling and executing such a program.
- The manner in which programs may be combined to form run-units.
- The elements of the language for which meaning is explicitly undefined.
- The elements of the language that is dependent on the capabilities of the processor.

2. Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of Elastic COBOL. At the time of publication, the editions indicated were valid. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 646:1991, Information technology - ISO 7-bit coded character set for information interchange.

ISO 1001:1986, Information processing - File structure and labeling of magnetic tapes for information interchange.

ISO 8601:1988, Data elements and interchange formats - Information interchange - Representation of dates and times.

ISO/IEC 9945-2:1993, Information technology - Portable Operating System Interface (POSIX) - Part 2: Shell and Utilities.

ISO/IEC 10646-1:1993, Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane.

ISO/IEC 10646-1:1993/Amd.1:1996, Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane, AMENDMENT 1: Transformation Format for 16 planes of group 00 (UTF-16).

ISO/IEC 10646-1:1993/Amd.2:1996, Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane, AMENDMENT 2: UCS Transformation Format 8 (UTF-8).

ISO/IEC CD 14651, International String Ordering - Method for Comparing Character Strings and Description of a Default Tailorable Ordering.

3. International Standard Conformance

Conforming implementation

Elastic COBOL conforms to ANSI X3.23-1985 at the highest level of compliance plus additional syntax and functionality.

Elastic COBOL does not support the optional Communications and Report Writer facilities.

Acceptance of standard language elements

There are general rules in standard COBOL that could have been classified as syntax rules. These rules are classified as general rules for the purpose of avoiding syntax checking, and do not reflect errors in standard COBOL. Elastic COBOL flags these errors on an individual basis.

Interaction with non-COBOL programs

Elastic COBOL provides a facility for directly interacting with Java classes on all systems and system programs accessible from the Java environment. On certain platforms, Elastic COBOL provides a facility for CALLing native programs written in C and COBOL.

Interaction between COBOL implementations

On certain platforms, Elastic COBOL provides a facility for CALLing native programs written in C and COBOL.

Implementer-defined language elements

Elastic COBOL includes additional language elements beyond standard COBOL.

Processor-dependent language elements

Processor refers to the entire computing system that is used to translate and execute source units, consisting of both hardware and relevant associated software. Language elements that pertain to specific processor components are listed in Processor-dependent language element list.

When support is claimed for a specific component, all language elements that pertain to that component are implemented.

Reserved words

Elastic COBOL recognizes as reserved words all reserved words of ANSI COBOL 1985. Elastic COBOL additionally recognizes its own extended reserved words.

Standard extensions

A standard extension is supported syntax for a facility defined in standard COBOL where support for that entire facility is not claimed.

Nonstandard extensions

Elastic COBOL includes language elements and functionality not defined in standard COBOL. Because of additional reserved words introduced, some compilation groups that meet the requirements of standard COBOL cannot be directly compiled. The 'wordlist' compiler option may be used to remove conflicting reserved words.

Substitute or additional language elements

Elastic COBOL does not require the inclusion of substitute or additional language elements in the compilation group in order to accomplish functionality identical to that of a standard COBOL language element.

Archaic language elements

Archaic language elements should not be used in new compilation groups because better programming practices exist. Because of widespread use in existing programs, there is no schedule for deleting archaic elements from standard COBOL; however, this may be reevaluated for any future revisions of standard COBOL.

Obsolete language elements

Obsolete language elements will be removed from the next revision of standard COBOL.

No language elements shall be deleted from the next revision of standard COBOL without having first been identified as obsolete.

Elastic COBOL can currently produce informational messages for items marked as obsolete in X/Open by use of the –FLAG:OBSX compiler option.

Externally provided functionality

Elastic COBOL requires additional runtimes from other vendors to operate certain extended functionality such as CICS client or MQSeries; these additional runtimes are mentioned in the appropriate sections. The Elastic COBOL runtime in combination with the standard Java runtime are sufficient for standard COBOL functionality and most extended Elastic COBOL functionality.

Limits

In general, standard COBOL specifies no upper limit on such things as the number of statements in a compilation group or the number of operands permitted in certain statements. Elastic COBOL does possess certain practical limits of these possibly limitless operations. These are documented in the appendix.

User documentation

Elastic COBOL satisfies the user documentation requirements specified for a conforming implementation. This documentation is the primary documentation source for Elastic COBOL.

Character substitution

The definition of the COBOL character set presents the complete COBOL character set for Elastic COBOL. Elastic COBOL additionally supports Unicode characters in the range of hex 0080-FF79 as acceptable COBOL characters.

A conforming compilation group

A conforming compilation group is one that does not violate the explicitly stated provisions and specifications of Elastic COBOL. In order for a compilation group to conform to standard COBOL, it shall not include any language elements not specified in this documentation.

Situations in which the results of executing a statement are explicitly undefined or unpredictable are identified in undefined language element list. A COBOL compilation group that allows these situations to happen is a conforming compilation group.

4. Description techniques

The techniques used to describe standard COBOL are:

- General formats
- Rules
- Arithmetic expressions
- Natural language text

General formats

General formats specify the syntax of the elements of Elastic COBOL and the sequence of arrangement of those elements.

The words, phrases, clauses, punctuation, and operands in each general format shall be written in the source program in the sequence given in the general format, unless otherwise specified by the rules of that format.

When more than one arrangement exists for a specific language construct, the general format is separated into multiple formats that are numbered and named.

Elements used in depicting general formats are:

- Keywords
- Optional words
- Operands
- Level numbers
- Options
- Brackets
- Braces
- Ellipses
- Punctuation
- Special characters
- Meta-terms that refer to other formats

Keywords

Keywords are reserved words or context-sensitive words. They are shown in uppercase and underlined in general formats. Certain punctuation keywords are not underlined where doing so would cause confusion.

Optional words

Optional words are reserved words or context-sensitive words. They are shown in uppercase and not underlined in general formats. They may be written to add clarity when the clause or phrase in which they are defined is written in the source unit.

The reserved words ‘IS’ and ‘ARE’ are always optional and may always be omitted.

Operands

An operand is an expression, a literal, or a reference to data. Operands are shown in lowercase and represent values or identification of items, conditions, or objects that the programmer supplies when writing the source unit.

The word 'integer' in a general format refers to an unsigned non-zero numeric literal.

Operands in general formats are suffixed with a number (-n) for unique reference in associated rules.

Level numbers

Specific level numbers appearing in general formats are required to be specified when the formats in which they appear are written in the source unit. Level number forms 01, 02, ..., 09, and 1, 2, ..., 9, respectively, may be written interchangeably.

Any number of 0’s may precede the level number.

Options

Options are indicated in a general format by vertically stacking alternative possibilities within brackets or braces or by listing the options with a pipe (|) choice character separating each choice option in the group. An option is selected by specifying one of the possibilities from a stack of alternative possibilities or by specifying a unique combination of possibilities from a series of brackets or braces.

Brackets

Brackets, [ ], enclosing a portion of a general format indicate that one of the alternatives contained within the brackets may be explicitly specified or that portion of the general format may be omitted. Brackets surround optional groups.

Braces

Braces, { }, enclosing a portion of a general format indicate that one of the alternatives contained within the braces shall be explicitly specified or is implicitly selected. If one of the alternatives contains only optional words, that alternative is

the default and is selected unless another alternative is explicitly specified. Braces surround required groups.

Ellipses

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

Given an ellipsis in a format, scanning right to left, determine the right bracket or right brace delimiter immediately to the left of the ellipsis; continue scanning right to left and determine the logically matching left bracket or left brace delimiter; the ellipsis applies to the portion of the format between the determined pair of delimiters.

In text other than general formats, the ellipsis (...) shows omission of a word or words when such omission does not impair comprehension. This is the conventional meaning of the ellipsis, and the use becomes apparent in context.

Punctuation

The separators comma and semicolon may be used anywhere the separator space is used in the formats. In the source program, these separators are interchangeable.

The separator period, when used in the formats, has the status of a required word.

Special characters

Special character words, punctuation characters, and separators that appear in formats, although not underlined, are required when such portions of the formats are used.

Meta-terms

Meta-terms appear in lowercase in general formats and are the names of subsections of general formats. Subsections are specified below the main format and are introduced by the phrase 'where x is:', with x replaced by the meta-term.

Rules

Syntax rules

Syntax rules supplement general formats and define or clarify the order in which words or elements are arranged to form larger elements such as phrases, clauses, or statements. Syntax rules may also either impose restrictions on individual words or elements or relax restrictions implied by words or elements.

These rules are used to define or clarify how the statement shall be written, i.e., the order of the elements of the statement and the restrictions or amplifications of what each element may represent.

General rules

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

Arguments

Argument rules specify requirements, constraints, or defaults associated with arguments to intrinsic functions.

Returned values

Returned value rules specify the semantics of an intrinsic function.

Arithmetic expressions

Some rules contain arithmetic expressions that specify part or all of the results of the COBOL syntax. In presenting the arithmetic expressions, the following additional notation, or different meaning for notation, is used.

Double subscripts

When a double subscript (term-jn) appears as an operand of an expression it refers to the n-th position/occurrence of term-j. Term-j will be locally substituted by an appropriate element of syntax.

Ellipses

Ellipses show that the number of terms and operators is variable.

Parentheses

Some arithmetic expressions contain one or more pairs of parentheses that do not change the order of evaluation. They are included for clarity.

Natural language text

A substantial portion of the COBOL specification is described in natural language. Syntax rules and semantic requirements may be included in the natural language description and are recognized by their context.

Hyphens in text

All hyphens appearing at the end of a line of text are the hyphens of meta-terms, keywords, optional words, or operands and are included in the term; otherwise, hyphens are not used to divide words at the end of a line.

5. Reference format

Reference format specifies the conventions for writing COBOL source programs, COBOL library text, and compiler directives. Elastic COBOL provides multiple reference formats, including fixed-form, free-form, and variable-form. Elastic COBOL attempts to automatically detect and use the appropriate format between fixed-form and free-form, but the format may be explicitly selected using compiler options.

Variable length is present for compatibility with some COBOL compilers and may only be selected using the compiler option; it is the same as fixed-form except that the right margin begins at column 16384. It is recommended not to use variable length reference format for new code.

When Elastic COBOL is automatically detecting the reference format, the COBOL source program and COBOL library text may be in different formats. Aside from using third-party libraries, the recommended course is to use the same format for both source program and library text.

Elastic COBOL is also capable of handling multiple file encodings, though only one file encoding may be used during a single compilation unit. Elastic COBOL supports ASCII, EBCDIC and Unicode (big- or little-endian) source code automatically on all platforms. This eases cross-platform development and remote editing.

The auto-detection method works as follows. If the first two characters are the Unicode directional markers, then Unicode is used. The first non-blank line is found, and then the first non-blank ASCII character is found. If the character is in column 7 or greater, fixed-form ASCII is used. If the character is an EBCDIC digit, EBCDIC fixed-form source code is used. If the character is an EBCDIC character capable of starting COBOL program source code, EBCDIC free-form source code is used. If the character is not an ASCII digit, free-form is used; otherwise fixed-form is used.

The following rules apply to the indicated reference formats:

1. **Free-form and fixed-form**

a. Reference format is described in terms of character positions on a line on an input-output medium.

b. Elastic COBOL accepts source programs written in reference format.

c. Elastic COBOL limits lines based on the carriage-return / linefeed pair or linefeed.

NOTE - The previous COBOL standard did not state what kinds of characters were used, but alphanumeric was generally assumed.

2. Fixed-form

a. Elastic COBOL processes fixed-form reference format lines as though the source program had been logically converted from fixed form to free form. (See Logical conversion.)

b. After logical conversion, the equivalent free-form source program shall meet the requirements of free-form reference format, except that all characters of the computer's character set shall be retained in alphanumeric literals and comments. (See rule 3b.)

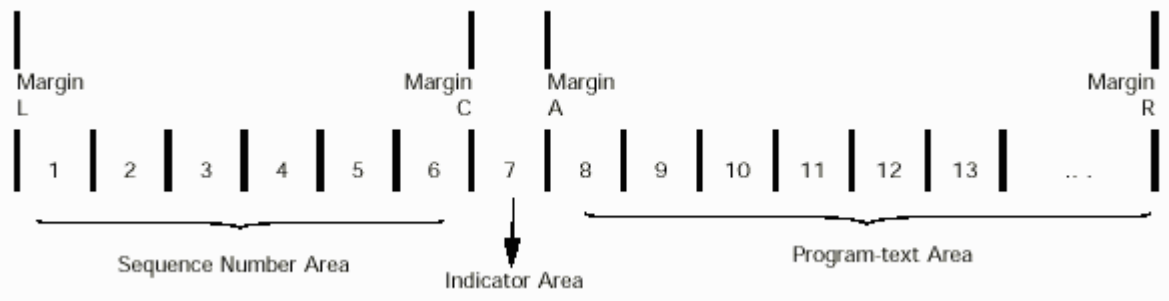
3. Free-form

a. The number of character positions on a line may vary from line to line, ranging from a minimum of 0 to a maximum of 255. This maximum is raised to 16383.

b. The carriage-return / linefeed pair or linefeed terminates a free-form line, including comment lines, and alphanumeric literals may contain ASCII characters 1-255 (based at 0).

Fixed-form reference format

The format of a fixed-form reference format line is depicted in figure 1, Fixed-form source reference format.



Fixed form source reference format

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

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

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

Margin R is immediately to the right of the rightmost character position of a line. The rightmost character position of a line is 72.

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

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

The program-text area begins in character position 8 and terminates with the character position immediately to the left of margin R.

Sequence numbers

The sequence number area may be used to label a source program line. The content of the sequence number area is defined by the user and may consist of any character in the computer's character set. There is no requirement that the content of the sequence number area appears in any particular sequence or be unique.

Continuation of lines

Any sentence, entry, phrase, or clause may be continued by starting subsequent line(s) in the program-text area. These subsequent lines are called the continuation line(s). The line being continued is called the continued line. Any COBOL word, literal, or PICTURE character-string may be broken in such a way that part of it appears on a continuation line. A compiler directive shall be contained entirely on one line. A compiler directive shall not be specified between the lines of a continued character string.

NOTE 1 - Continuation of COBOL words is an archaic feature and its use should be avoided

A hyphen in the indicator area of a line indicates that the first nonblank character in the program-text area of the current line is the successor of the last nonblank character of the preceding line, excluding intervening comment lines or blank lines without any intervening space. However, if the continued line contains an alphanumeric literal without a closing delimiter, the first nonblank character in the program-text area of the continuation line shall be a quotation symbol matching the quotation symbol used in the opening delimiter. The continuation starts with the character immediately after that quotation symbol. All spaces at the end of the continued line are considered part of the literal.

If there is no hyphen in the indicator area of a line, it is assumed that the first nonblank character in the line is preceded by a space.

All characters composing any two-character or three-character separator shall be specified on the same line.

National literals may only be continued using concatenation. (See continuation lines for information about multiple lines).

NOTE 2 - Literals may be concatenated to achieve continuation.

Blank lines

A blank line is one that is blank from margin C to margin R, inclusive. A blank line may appear anywhere in the source program. (See Continuation of lines.)

A blank line for purposes of automatic detection of source format is one that is blank in all character positions.

Comments

A comment may be a comment line or an in-line comment.

Any combination of characters from the computer's character set may be included in a comment.

Comments shall be listed if a listing is being produced, but shall serve only as documentation and shall not be checked syntactically.

Comments may include Unicode characters or other DBCS characters.

Comment lines

A comment line is any line with an asterisk or slant in the indicator area of the line. A comment line may be written as any line in a compilation-group or as any line in library text. The slant in the indicator area causes page ejection prior to printing the comment line in the listing of the source program; an asterisk in the indicator area causes printing of the line at the next available line position in the listing.

In-line comments

The two contiguous COBOL characters `*>' or the character ‘|’ is a comment indicator. An in-line comment consists of a comment indicator followed by all successive character positions on the same line up to margin R, inclusive. An in-line comment may be written in the program-text area on any line of a compilation-group or library text except on a compiler directive line, wherever a separator space may be written.

For purposes of evaluating library text, pseudo-text, and source text, an in-line comment shall have the value of a separator space.

Block Comments

A block comment is a comment which starts on one line and which may continue for multiple lines, including the other forms of comments. Its typical use is to block out a section of code. A block comment must begin and end in Area A or Area B, not the indicator area. The beginning of a block comment is marked using the two contiguous characters '/*' and the ending of a block comment is marked using the two contiguous characters '/*'.

Debugging lines

A debugging line is any line with the debugging symbol, character 'D' or 'd', in the indicator area of the line. Any debugging line that consists solely of spaces from margin A to margin R is considered the same as a blank line. Successive debugging lines may be specified.

A debugging line is permitted any place in the source unit after the SOURCE-COMPUTER paragraph.

A debugging line is permitted at any point in the source unit, including before the IDENTIFICATION DIVISION.

In-line debugging

A debugging line may optionally marked in Area or Area B by using the three contiguous characters '>>D'.

Debugging Line Notes

A debugging line is an obsolete element in this draft International Standard and is to be deleted from the next revision of standard COBOL.

Free-form reference format

In free-form reference format, the text of the source program can be written anywhere on a line, except that there are specific rules for comments, debugging lines, and continuation.

Continuation of lines

Continuation of lines is not permitted in free-form. Continuation of nonnumeric literals is permitted using concatenation (&).

Blank lines

A blank line is one that contains nothing but space characters or is a line with zero character positions. A blank line may appear anywhere in a compilation-group or library text.

Blank lines interspersed among lines containing the parts of a literal shall be listed if a listing is being produced, but shall have no effect on the meaning or compilation of the literal.

Comments

The two contiguous COBOL characters '*>' or the character ‘|’ is a comment indicator. A comment consists of a comment indicator followed by all successive character positions on the line. A comment terminates at the end of the line on which the comment is written.

Any combination of characters from the computer's character set may be included in a comment, except as indicated in Reference format, rule 3b.

Comments shall be listed if a listing is being produced, but shall serve only as documentation and shall not be checked syntactically.

A comment may be a comment line or an in-line comment.

Comment lines

A comment preceded exclusively by zero, one, or more spaces on the line on which the comment is written is a comment line. A comment line may be written as any line in a compilation-group or as any line in library text.

Comment lines interspersed among lines containing the parts of a literal shall be listed if a listing is being produced, but shall have no effect on the meaning or compilation of the literal.

In-line comments

A comment preceded on a line by one or more COBOL words or character-strings immediately followed by one or more separator spaces is an in-line comment. An in-line comment may be written wherever a separator space may be written on any line of a compilation-group or library text except on a compiler directive line.

For purposes of evaluating library text, pseudo-text, and source text, an in-line comment shall have the value of a separator space.

Block Comments

A block comment is a comment which starts on one line and which may continue for multiple lines, including the other forms of comments. Its typical use is to block out a section of code. A block comment must begin and end in Area A or Area B, not the indicator area. The beginning of a block comment is marked using the two contiguous characters '/*' and the ending of a block comment is marked using the two contiguous characters '/*'.

Debugging lines

The three contiguous COBOL characters '>>D' immediately followed by a space are a debugging indicator. A debugging line consists of a debugging indicator, optionally preceded by one or more space characters, followed by all successive character positions on the line. A debugging line is terminated at the end of the line.

A ‘D’ or ‘d’ followed by a space at the beginning of a free format line shall indicate a debugging line.

A debugging line may be written as any line in a source unit after the SOURCE-COMPUTER paragraph. Successive debugging lines may be specified.

After all COPY and REPLACE statements have been processed, a debugging line shall have the characteristics of a comment line if the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph.

The contents of a debugging line shall be such that a syntactically correct program is formed with or without the debugging line.

In-line debugging

A debugging line may optionally marked in Area or Area B by using the three contiguous characters '>>D'.

Debugging Line Notes

A debugging line is an obsolete element in this draft International Standard and is to be deleted from the next revision of standard COBOL.

Logical conversion

Source text and library text in fixed-form reference format shall be logically converted at compile time to equivalent free-form reference format, without the restriction of maximum line length.

NOTE - Fixed-form reference format is logically converted during compilation to free-form to simplify understanding of other rules of the language, for example, the COPY statement with the REPLACING phrase and the REPLACE statement.

Logical conversion shall have the effect of taking place after the processing of the copy action of a COPY statement and before the replacing action of a COPY statement with the REPLACING phrase or of a REPLACE statement.

The rules given below shall be applied logically for each fixed-form line to produce a free-form line. For purposes of these rules, the character positions contained between margin A and margin R of a fixed-form source line are designated as "program-text" and retain this designation in the equivalent free-form source line.

1. If the indicator area of the fixed-form line contains a character 'D' or 'd', the equivalent free-form line shall contain a debugging indicator in the first four character positions, followed by the program-text from the fixed-form line.
2. If the indicator area of the fixed-form line contains a hyphen, the equivalent free-form line shall contain a space in character position one followed by the program-text from the fixed-form line. For purposes of this conversion, this free-form source line is a continuation line and the closest preceding free-form line that is not a comment line, a blank line, or a compiler directive is the continued line. Free-form continuation is completed as follows:

a. If the last character-string in the free-form continued line begins an alphanumeric or national literal that is not terminated by a closing delimiter, the free-form program-text in the continued line is appended with a closing delimiter, immediately followed by a hyphen.

b. Otherwise, the first character-string in the free-form continuation line is appended, with no intervening space, to the last character-string in the program-text of the free-form continued line. The first character-string in the free-form continuation line is replaced with blanks.

3. If the indicator area of the fixed-form line contains an asterisk (*), the equivalent free-form line shall contain a comment indicator in the first two character positions followed by the program-text from the fixed-form line.
4. If the indicator area of the fixed-form line contains a slash (/), the equivalent free-form line shall contain a PAGE compiler directive specifying as commentary the program-text from the fixed-form line.
5. Otherwise, the equivalent free-form line shall contain the program-text from the fixed-form line.

Thereafter, compilation continues with the logically equivalent free-form source.

6. Compiler directing facility

The compiler directing facility controls the processing of a compilation group. It consists of compiler directing statements for specifying source text manipulation and compiler directives for specifying compilation options.

Source text manipulation

Source text manipulation provides a facility to insert and replace source text as part of the compilation of a compilation group. Source text manipulation involves compiler directing statements and sentences, library text available to the compiler at compile time, and source text.

Source text manipulation elements

Language elements referenced and not defined in Source text manipulation have the meaning defined in Language fundamentals.

Compiler directing statements and sentences

A compiler directing statement instructs the compiler to take a specific action during compilation. Compiler directing statements in the source text manipulation facility are the COPY statement, REPLACE statement and preprocessing statements terminated by a separator period.

Source text and library text

Source text is the primary input to the compiler for a single compilation group. Library text is secondary input to the compiler as a result of processing a COPY statement.

Source text manipulation processes source text and library text as a sequence of one or more character-strings, separators, and comments. Source text and library text enclosed within pseudo-text delimiters are processed as pseudo-text.

Pseudo-text

Pseudo-text may be any sequence of zero or more text-words, comments, and the separator space bounded by, but not including, pseudo-text delimiters.

Text-words

A text-word is a character or a sequence of contiguous characters in COBOL source text, library text, or pseudo-text that forms any of the following:

1. a separator, except for: a space; a pseudo-text delimiter; the opening and closing delimiters for alphanumeric, Boolean, and national literals; and the right and left parentheses, regardless of their context; or
2. an alphanumeric, Boolean, or national literal including the opening and closing delimiters that bound the literal; or

3. any other sequence of contiguous COBOL characters bounded by separators, except for: comments and the word ‘COPY’.

COPY statement

The COPY statement incorporates library text into a COBOL compilation group.

General format

Format 1 :

```
COPY { literal-1 | text-name-1 } [NOLIST] [SUPPRESS] [ {OF|IN}
   {literal-2 | library-name-1}] [NOLIST] [SUPPRESS]
        [ REPLACING   [LEADING|TRAILING|INDEPENDENT]
   { {==pseudo-text-1== | identifier-1 | literal-3 | word-1 }
        BY {==pseudo-text-2 | identifier-2 | literal-4 | word-2} } … ]
```

Format 2:

```
COPY RESOURCE { literal-1 | text-name-1 } .
```

Syntax rules

Format 1 :

1. A COPY statement may be specified anywhere in a compilation group that a character string or a separator, other than the closing delimiter of a literal, may appear except that a COPY statement shall not appear within a COPY statement.
2. A COPY statement shall be preceded by a space except when it is the first statement in a compilation group.
3. A COPY statement shall be terminated by a separator period.
4. Within one COBOL library, each text-name shall be unique.
5. Literal-1 and literal-2 shall be alphanumeric literals and shall not be figurative constants. Literal-1 shall refer to a filename offset from the current directory, or offset from the directory specified by literal-2. Literal-1 and literal-2 shall be specified according to the native filename, i.e., using ‘\’ as the separator in Windows, ‘/’ in Posix, etc. Text-name-1 is treated as literal-1 and
6. Text-name-1 or literal-1 identifies the library text to be processed by the COPY statement.
7. Pseudo-text-1 shall contain one or more text-words.
8. Pseudo-text-2 shall contain zero, one, or more text-words.
9. Character-strings within pseudo-text-1 and pseudo-text-2 may be continued in accordance with the rules of reference format.
10. Word-1 or word-2 may be any single COBOL word except 'COPY'.
11. Elastic COBOL allows a match buffer of 16384 bytes.
12. Pseudo-text-1 shall not consist entirely of a separator comma or a separator semicolon.

13. Pseudo-text-1 and pseudo-text-2 may contain debugging lines. 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.

Format 2 :

1. literal-1 or text-name-1 refers to a resource file such as an image or audio clip to be included. This statement is treated as commentary since resources must be included separately, so the name is not checked for validity.

General rules

1. The compilation of a compilation group containing COPY statements is logically equivalent to processing all COPY statements prior to the processing of the resultant compilation group.
2. The syntactic correctness of library text cannot be independently determined. Except for COPY and REPLACE statements, the syntactic correctness of a COBOL source unit cannot be determined until all applicable COPY and REPLACE statements have been completely processed.
3. Library text shall conform to the rules for COBOL reference format.
4. Debugging lines are permitted within library text.
5. The effect of processing a COPY statement is that the library text associated with text-name-1 or the value of literal-1 is copied into the source text, logically replacing the entire COPY statement beginning with the reserved word COPY and ending with the separator period, inclusive.
6. If library-name-1 or literal-2 is unspecified, Elastic COBOL searches along the path specified by –lib, if any, or the current PATH environment variable.
7. If the REPLACING phrase is not specified, the library text is copied unchanged.
8. If the REPLACING phrase is specified, the library text is copied and each matched occurrence of pseudo-text-1, identifier-1, word-1, or literal-3 in the library text is replaced by the corresponding pseudo-text-2, identifier-2, word-2 or literal-4.
9. For purposes of matching, identifier-1, word-1, and literal-3 are treated as pseudo-text containing only identifier-1, word-1, or literal-3, respectively.
10. The comparison operation to determine text replacement occurs in the following manner:
    a. The leftmost library text-word that is not a separator comma or a separator semicolon is the first text-word used for comparison. Any text-word or space preceding this text-word is copied into the source text. Starting with the first text-word for comparison and first pseudo-text-1, identifier-1, word-1, literal-3, that was specified in the REPLACING phrase, the entire REPLACING phrase operand that precedes the reserved word BY is compared to an equivalent number of contiguous library text-words.
    b. Pseudo-text-1, identifier-1, word-1, or literal-3 matches the library text only if the ordered sequence of text-words that forms pseudo-text-1, identifier-1, word-1, or literal-3 is equal, character for character, to the ordered sequence of library text-words.

c. The following rules apply for the purpose of matching:

- Each occurrence of a separator comma, semicolon, or space in pseudo-text-1 or in the library text is considered to be a single space. Each sequence of one or more space separators is considered to be a single space.
- Except for the content of literals, each alphanumeric character is equivalent to its corresponding national character and each lowercase letter is equivalent to its corresponding uppercase letter, as specified for the COBOL character set in COBOL character set.

NOTE - A concatenation expression, for example "A" & "B", is recognized as three text-words for purposes of matching.

- A compiler directive line in source text shall be considered a single text-word that does not match any text-word within pseudo-text-1 or partial-word-1.
- A debugging indicator shall be treated as if it did not appear in the line. Text-words within a debugging line participate in the matching.
- Comments or blank lines occurring in the source text and in pseudo-text-1 are ignored.

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

e. When all the REPLACING phrase operands have been compared and no match has occurred, the leftmost library text-word is copied into the source text. The next successive library text-word is then considered as the leftmost library text-word, and the comparison cycle starts again with the first pseudo-text-1, identifier-1, word-1, literal-3, or partial-word-1 specified in the REPLACING phrase.

f. When a match occurs between pseudo-text-1, identifier-1, word-1, or literal-3 and the library text, the corresponding pseudo-text-2, identifier-2, word-2, or literal-4 is placed into the source text. When a match occurs between partial-word-1 and the library text-word, the library text-word is placed into the source text with the matched characters either replaced by partial-word-2 or deleted when partial-word-2 consists of zero text-words. The library text-word immediately following the rightmost text-word that participated in the match is then considered as the leftmost text-word. The comparison cycle starts again with the first pseudo-text-1, identifier-1, word-1, literal-3, or partial-word-1 specified in the REPLACING phrase.

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

11. If the REPLACING phrase is specified, the library text shall not contain a COPY statement and the source text that results from processing the REPLACING phrase shall not contain a COPY statement.

12. The source text that results from the processing of a COPY statement shall be in logical free-form reference format. For purposes of compilation, text-words after replacement are placed in the resultant source text in accordance with the rules of free-form reference format. When copying text-words of pseudo-text-2 into the source text, additional spaces may be introduced only between text-words where there already exists a space or a space is assumed.

    NOTE - A space is assumed at the end of a source line.

13. Comments or blank lines appearing in pseudo-text-2 are copied into the source text unchanged whenever pseudo-text-2 is placed into the source text as a result of text replacement. Comments or blank lines appearing in library text are copied into the source text unchanged with the following exception: a comment or blank line in library text is not copied if that comment or blank line appears within the sequence of text-words that match pseudo-text-1.

14. If additional lines are introduced into the source text as a result of a COPY statement, each text-word introduced appears on a debugging line if the COPY statement begins on a debugging line or if the text-word being introduced appears on a debugging line in library text. When a text-word specified in the BY phrase is introduced, it appears on a debugging line if the first library text-word being replaced is specified on a debugging line. Except in the preceding cases, only those text-words that are specified on debugging lines where the debugging line is within pseudo-text-2 appear on debugging lines in the resultant source text. If any literal specified as literal-4 or within pseudo-text-2 or library text is of a length too great to be accommodated on a single line without continuation to another line in the resultant source text and the literal is not being placed on a debugging line, additional continuation lines are introduced which contain the remainder of the literal. If replacement requires that the continued literal be continued on a debugging line, the program is in error.

15. If additional lines are introduced into the source text as the result of processing a REPLACING phrase of a COPY statement, the indicator of the introduced lines shall be the same as the indicator appearing on the line at which the replaced text begins.

16. If the REPLACING phrase is not specified, the library text may contain a COPY statement that does not include a REPLACING phrase. Recursive copying of library text is not permitted; that is, the library text being copied shall not cause the processing of a COPY statement that directly or indirectly copies itself. Elastic COBOL supports 256 open files where this is supported by the operating system; this open limit is shared with the main source files.

17. NOLIST or SUPPRESS is used to suppress the creation of a listing file for the COPY contents. This is treated as commentary.

18. LEADING ensures that the REPLACING from text must be preceded by white space.

19. TRAILING ensures that the REPLACING from text must be followed by white space.

20. INDEPENDENT that the REPLACING from text must be preceded and followed by white space.

REPLACE statement

The REPLACE statement is used to replace source text in a separately compiled COBOL source unit.

General format

Format 1 (replacing):

REPLACE [LEADING|TRAILING|INDEPENDENT] {==pseudo-text-1== BY ==pseudo-text-2==} …

Format 2 (off):

REPLACE OFF

Syntax rules

1. A REPLACE statement may be specified anywhere in a compilation group that a character-string or a separator, other than a closing delimiter of a literal, may appear.
2. A REPLACE statement shall be preceded by a space except when it is the first statement in a compilation group.
3. A REPLACE statement shall be terminated by a separator period.
4. Pseudo-text-1 shall contain one or more text-words.
5. Pseudo-text-2 shall contain zero, one, or more text-words.
6. Character-strings within pseudo-text-1 and pseudo-text-2 may be continued in accordance with the rules of reference format.
7. Elastic COBOL allows a match buffer of 16384 bytes.
8. Pseudo-text-1 shall not consist entirely of a separator comma or a separator semicolon.
9. Pseudo-text-1 and pseudo-text-2 may contain debugging lines.
10. Compiler directive lines shall not be specified within pseudo-text-1 or pseudo-text.

General rules

1. Except for COPY and REPLACE statements, the syntactic correctness of a source unit cannot be determined until after all applicable COPY and REPLACE statements have been completely processed.
2. Pseudo-text-1 specifies the source text to be replaced by pseudo-text-2.
3. A given REPLACE statement is in effect from the point at which it is specified until the next occurrence of a REPLACE statement without the ALSO phrase or the end of the separately compiled program is reached.
4. Any REPLACE statements contained in a source unit are processed after the processing of any COPY statements contained in that source unit.
5. The comparison operation to determine text replacement occurs in the following manner:

a. Starting with the leftmost source text-word and the first pseudo-text-1 or partial-word-1, pseudo-text-1 or partial-word-1 is compared to an equivalent number of contiguous source text-words.

b. Pseudo-text-1 matches the source text if, and only if, the ordered sequence of text-words that forms pseudo-text-1 is equal, character for character, to the ordered sequence of source text-words.

c. The following rules apply for the purpose of matching:

- Each occurrence of a separator comma, semicolon, or space in pseudo-text-1 or in the source text is considered to be a single space. Each sequence of one or more space separators is considered to be a single space.

- Except for the content of literals, each alphanumeric character is equivalent to its corresponding national character and each lowercase letter is equivalent to its corresponding uppercase letter, as specified for the COBOL character set in COBOL character set.

NOTE - A concatenation expression, for example "A" & "B", is recognized as three text-words for purposes of matching.

- A compiler directive line in source text shall be considered a single text-word that does not match any text-word within pseudo-text-1 or partial-word-1.

- A debugging indicator shall be treated as if it did not appear in the line.

- Comments or blank lines occurring in source text and in pseudo-text-1 are ignored.

a. If no match occurs, the comparison is repeated with each next successive occurrence of pseudo-text-1, until either a match is found or there is no next successive occurrence of pseudo-text-1.

b. When all occurrences of pseudo-text-1 have been compared and no match has occurred, the next successive source program text-word is then considered as the leftmost source program text-word, and the comparison cycle starts again with the first occurrence of pseudo-text-1.

c. When a match occurs between pseudo-text-1 and the source text, the corresponding pseudo-text-2 replaces the matched text in the source text. The source text-word immediately following the rightmost text-word that participated in the match is then considered as the leftmost source text-word. The comparison cycle starts again with the first occurrence of pseudo-text-1.

d. The comparison operation continues until the rightmost text-word in the source text that is within the scope of the REPLACE statement has either participated in a match or been considered as a leftmost source text-word and participated in a complete comparison cycle.

6. The text produced as a result of the processing of a REPLACE statement shall contain neither a COPY statement nor a REPLACE statement.

7. The source text that results from the processing of a REPLACE statement shall be in logical free-form reference format. Text-words inserted into the source text as a result of processing a REPLACE statement are placed in accordance with the rules of free-form reference format. When inserting text-words of pseudo-text-2 into the source text, additional spaces may be introduced only between text-words where there already exists a space or a space is assumed.

   NOTE - A space is assumed at the end of a source line.

8. Comments or blank lines in pseudo-text-2 are placed into the source text unchanged whenever pseudo-text-2 is placed into the source text as a result of text replacement. A comment or a blank line in source text shall not be placed into the resultant source text if that comment or blank line appears within the sequence of text-words that match pseudo-text-1.
9. If a REPLACE statement is specified on a debugging line, the source text that is the result of processing the REPLACE statement shall be on a debugging line.
10. If any literal within pseudo-text-2 is of a length too great to be accommodated on a single line without continuation to another line in the resultant source text and the literal is not being placed on a debugging line, additional continuation lines are introduced which contain the remainder of the literal. If replacement requires the continued literal to be continued on a debugging line, the program is in error.
11. The effect of a format 2 REPLACE statement is to cancel all format 1 REPLACE statements currently in effect.
12. LEADING ensures that the REPLACING from text must be preceded by white space.
13. TRAILING ensures that the REPLACING from text must be followed by white space.
14. INDEPENDENT that the REPLACING from text must be preceded and followed by white space.

Compiler directives

A compiler directive specifies options or constants for use by the compiler.

General format

$command [parameters]

Syntax rules

1. A compiler directive shall be specified on one line, except for the IF directives for which specific rules are specified.
2. A compiler directive shall have its $ in the indicator column.
3. A compiler-directive word is reserved within the context of the compiler directive in which it is specified and may be used elsewhere as any type of COBOL word.
4. A compiler directive may be specified anywhere in a compilation-group except
   a. as restricted by the rules for the specific compiler directive,
   b. within a source text manipulation statement,

c. between the lines of a continued character string,

5. Compiler directive lines may be specified within library text.

General rules

1. A compiler directive shall be treated as a single blank line during the matching operation of a COPY or REPLACE statement. A directive will not match any pseudo-text or partial-word and therefore will not be affected by the replacing action.
2. A compiler directive shall be processed before, during, and after the processing of COPY and REPLACE statements as indicated in the specific rules for each directive.

Conditional compilation

The use of certain compiler directives provides a means of including or omitting selected lines of source code. This is called conditional compilation. The compiler directives that are used for conditional compilation are 78 level constants and the IF directive. The 78 level constants are used to define constants that are referenced in the IF directive in order to select lines of code that are to be compiled or are to be omitted during compilation.

Constant conditional expression

A constant conditional expression is a conditional expression in which all the operands are literals or arithmetic expressions containing only literal terms. A special form of condition known as a defined condition may also be used as part of a constant conditional expression.

Syntax rules

A constant conditional expression shall be one of the following:

1. A simple relation condition in which both sides are either literals or arithmetic expressions containing only literal terms;
2. A defined condition;
3. A complex condition formed by combining the above two forms of simple conditions into complex conditions as described in 8.8.4.2, Complex conditions. Abbreviated combined relation conditions are not allowed.

General rules

Any arithmetic expression specified in a constant conditional expression is evaluated following the normal precedence rules. After each operation the result is

truncated to the integer part of the value. The resultant value is considered to be an integer.

Defined condition

General format

Format 1:

Identifier-name-1 [NOT] DEFINED

Format 2:

Constant-name-1 [NOT] {< | > | =} literal-1

General rules

Format 1:

1. A defined condition using the DEFINED syntax evaluates TRUE if identifier-name-1 has previously been defined.
2. A defined condition using the NOT DEFINED syntax evaluates TRUE if identifier-name-1 has not previously been defined.

Format 2:

1. Constant-name-1 must have been previously defined as a 78-level constant. The type is integer is given an integer value; the type is alphanumeric otherwise.
2. Constant-name-1 is compared using integer or alphanumeric comparison to literal-1. The expression evaluates as true depending on the comparison between constant-name-1's contents and literal-1 using the operator <, >, =, NOT <, NOT >, or NOT =.

IF directive

The IF directive provides for 1 or 2-way conditional compilation.

General format

```
$IF constant-conditional-expression-1
        source-lines-1
[$ELSE
        source-lines-2
]
$ENDIF
```

Syntax rules

1. Source-lines-1 and source-lines-2 shall be specified on a new line preceded only by zero, one or more space characters. The $ shall be in the indicator column for each line.

2. Source-lines-1 and source-lines-2 may be any kind of source lines, including directives

General rules

1. The IF directive is processed during the processing of COPY and REPLACE statements.
2. If constant-conditional-expression-1 evaluates to TRUE, then source-lines-1 are compiled as part of the COBOL source program, and source-lines-2 are ignored.
3. If constant-conditional-expression-1 evaluates to FALSE, the source-lines-1 are ignored and source-lines-2, if specified, are compiled as part of the COBOL source program.

Message Directives

The message directives give the user feedback during compilation. This is most useful for conditional compilation.

General format

Format 1:
$DISPLAY text

Format 2:
$WARNING text

Format 3:
$ERROR text

Format 4:
$INFO text

General rules

1. The $ must be in the indicator column immediately followed by the statement.
2. The text can be any text to display to the user.

Format 1:

The text is displayed to the user during compilation with no further result.

Format 2:

A warning message formed from text is displayed to the user during compilation.

Format 3:

An error message formed from text is displayed to the user during compilation.

Format 4:

An info message formed from text is displayed to the user during compilation.

Set Directives

The SET directive is used to set various compiler options from within the source code.

General format

Format 1:

$SET SOURCEFORMAT“FREE”

$SET SOURCEFORMAT”FIXED”

$SET SOURCEFORMAT”VARIABLE”

Format 2:

$SET OPTIONAL-FILE

$SET NOOPTIONAL-FILE

Format 3:

$SET STDERR

$SET NOSTDERR

Format 4:

$SET DEFAULTBYTE “integer-1”

Format 5:

$SET ANIM

$SET NOANIM

Format 6:

$SET APOST

$SET QUOTE

Format 7:

$SET BOUND

Format 8:

$DATATYPE old-type=new-type

where new-type or old-type is:

COMPUTATIONAL-1-REV

COMPUTATIONAL-2-REV

COMPUTATIONAL-1-MVS

COMPUTATIONAL-2-MVS

COMPUTATIONAL-5-REV

COMPUTATIONAL-X-REV

PACKED-DECIMAL-I

PACKED-DECIMAL-A

COMPUTATIONAL-1

COMPUTATIONAL-2

COMPUTATIONAL-3

COMPUTATIONAL-4

COMPUTATIONAL-5

COMPUTATIONAL-6

COMPUTATIONAL-X
COMPUTATIONAL-N
COMPUTATIONAL-S
COMPUTATIONAL-D
PACKED-DECIMAL
UNSIGNED-SHORT
COMPUTATIONAL
UNSIGNED-LONG
UNSIGNED-INT
SIGNED-SHORT
SIGNED-LONG
SIGNED-INT
COMP-1-REV
COMP-2-REV
COMP-1-MVS
COMP-2-MVS
BINARY-REV
COMP-5-REV
COMP-X-REV
BINARY
DOUBLE
COMP-1
COMP-2
COMP-3
COMP-4
COMP-5
COMP-6
COMP-S
COMP-D
COMP-X
COMP-N
FLOAT
COMP

Format 9:
$XFD

Format 10:
$FLEXUS or $MICROFOCUS

Format 11:
$ACUGFX or $ELASTIC COBOLGFX

Format 12:
$ACUCOBOL

Format 13:
$TRUNC(BIN)

Format 14:
$COMMENT text

Format 15:
$COPYRIGHT text

Format 16:
$PAGE text

Format 17:
$TITLE text

Format 18:
$VERSION text

Format 19:
$INCLUDE copy-text

Format 20:
$SET KEYCOMPRESS value
$SET NOKEYCOMPRESS

Format 21:
$SET IDXFORMAT value

Format 22:
$SET DATACOMPRESS value
$SET NODATACOMPRESS

Format 23:
$SET STICKY-PERFORM
$SET NOSTICKY-PERFORM

Format 24:
$SET ALTER
$SET NOALTER

General rules

The $ must be in the indicator column.

Format 1:

1. The SOURCE FORMAT directive indicates that the source text or library text following the directive and continuing through a subsequent SOURCE FORMAT directive shall be treated as fixed form if FIXED is specified, or as free form if FREE is specified, or variable form if VARIABLE is specified. (See 6.1, Fixed-form reference format, and 6.2, Free-form reference format.)
2. The default reference format of a compilation-group is fixed form.
3. The default reference format of library text is the reference format that was in effect for the terminating period of the COPY statement that resulted in processing of this library text.

4. If a SOURCEFORMAT directive is specified in library text, the specified format shall be in effect until another SOURCEFORMAT directive is encountered or the end of the library text is reached. When the processing of that library text is completed, the reference format shall revert to the format in effect for the terminating period of the COPY statement that resulted in processing of that library text.

Format 2:

1. If OPTIONAL-FILE is specified, all file assignments from that point onwards are treated as if OPTIONAL were specified.
2. If NOOPTIONAL-FILE is specified, all file assignment from that point onwards is treated in standard fashion, using OPTIONAL only if specified.

Format 3:

1. STDERR sets the compiler error output to be standard error.
2. NOSTDERR sets the compiler error output to be standard output.

Format 4:

DEFAULTBYTE sets the default byte with which to fill memory. Integer-1 is the ASCII value (0-255) to use. The default is 0. The most common alternative is 32 (space). To ensure that this directive takes effect, use it before the identification division, or use the command line compiler option.

Format 5:

ANIM enables debug mode, NOANIM disables debug mode.

Format 6:

APOST sets the QUOTES figurative constant to be the apostrophe (‘). QUOTE sets the QUOTES figurative constant to be the double quotation marks (“).

Format 7:

BOUND tells Elastic COBOL to do bounds checking on table accesses; this is the default and BOUND is treated as commentary.

Format 8:

In all text compiled after the $DATATYPE directive, all references to usage old-type will be automatically replaced by references to new-type. This allows effective usage of datatypes for compatibility with other COBOL implementations. Several DATATYPE directives are implied by the data compatibility compiler options.

Format 9:

$XFD is ignored as commentary.

Format 10:

Implies the compiler option -Dcm for Micro Focus compatibility.

Format 11:

Override automatic detection of desired graphics to be compatible with AcuCOBOL or original Elastic COBOL graphical syntax. Generally allow automatic detection to handle the keyword selections automatically.

Format 12:

Enable all AcuCOBOL compatibility, implying the -Dca compiler option.

Format 13:

Implies compiler option -Db:trunc, allowing BINARY usage items to be truncated at the natural binary division according to the number of bytes used rather than the picture.

Format 14:

Embed comment 'text' in generated source and listing file if present.

Format 15:

Embed copyright 'text' in generated source and listing file if present.

Format 16:

Embed page split marked 'text' in generated source and listing file if present.

Format 17:

Embed title marked 'text' in generated source and listing file if present.

Format 18:

Embed version marked 'text' in generated source and listing file if present.

Format 19:

Include the given text file in the source as if by COPY.

Format 20:

Set the keycompression value or disable keycompression. This value is made available to the indexed file runtimes, but may only be used in certain file systems.

Format 21:

Set the idxformat value. This value is made available to the indexed file runtimes, but may only be used in certain file systems.

Format 22:

Set the datacompress value or disable datacompress. This value is made available to the indexed file runtimes, but may only be used in certain file systems.

Format 23:

STICKY-PERFORM is the ANSI described manner of PERFORM, but not the manner of PERFORM that most programmers expect; its use is discouraged. NOSTICKY-PERFORM allows more flexibility in the usage of PERFORM and is the form of PERFORM most programmers expect, allowing recursive PERFORM calls.

Format 24:

Set whether the ALTER verb is allowed in the program. Certain optimizations may be made if there is a guarantee that no ALTER will be done.

7. Language fundamentals

Character sets

The basic unit of the COBOL language specification is the character. COBOL syntax is described using the COBOL character set, which is a set of characters independent of their encoding. Comments and the content of certain classes of literals in the source program and the content of certain classes of data items are represented in a coded character set -- the computer's character set. Additional coded character sets, called alphabets, may be defined in a source program for representation of data on external media.

Computer's character set

The computer's character set is used to represent comments and the content of alphanumeric and national literals in the source program and to represent the content of alphabetic, alphanumeric, and national data items.

The computer's character set at execution time is also referred to as the native character set. A computer shall have a native alphanumeric character set and a native national character set. Elastic COBOL's native alphanumeric character set is ASCII; source code on EBCDIC machines is stored as EBCDIC. Elastic COBOL's national character set is Unicode.

COBOL character set

The COBOL character set is used to represent COBOL words, separators, and the content of hexadecimal literals in a COBOL source program. The COBOL character set consists of the basic letters, basic digits, basic special characters, and extended letters as shown in table 1, COBOL character set.

COBOL character set

Description	Character	Meaning
Basic letters	A, B, ... Z	upper-case letters in ISO/IEC 646
	a, b, ... z	lower-case letters in ISO/IEC 646
Basic digits	0, 1, ... 9	Digits
Basic special character		Space
	+	plus sign
	-	minus sign (hyphen)
	*	Asterisk
	/	slant (slash, solidus)
	=	equal sign
	$	dollar sign
	,	Comma
	;	Semicolon
	.	period (decimal point, full stop)
	"	quotation mark
	'	Apostrophe
	(	left parenthesis
	)	right parenthesis
	>	greater than
	<	less than
	&	Ampersand
	:	Colon
	_	Underscore

Extended letters, additional characters used to form user-defined words, are available from the Unicode set of hex 0080 through FF7F.

General rules

1. The mapping of source characters is ASCII on ASCII machines, EBCDIC on EBCDIC machines, and Unicode when the source code has the Unicode direction marker at in the first two bytes of the file.
2. Within a compilation group, COBOL characters may be represented in either an alphanumeric coded character set or a national coded character set or both. The following rules apply:
   a. Except when used in some alphanumeric literal formats and except for some picture symbols, each lowercase COBOL basic letter is equivalent to its corresponding uppercase COBOL basic letter, if any.
   b. Each basic letter, basic digit, and basic special character represented in the alphanumeric character set is equivalent to its corresponding basic letter, basic digit, and basic special character represented in the national character set, respectively.
3. Each Unicode national character possesses lowercase-uppercase equivalence by converting internally to uppercase where suitable in the represented national language.

   This is active for the following Unicode regions:

   a. Latin

b. Latin-1 Supplement
c. Latin Extended-A
d. Latin Extended-B
e. Greek
f. Coptic Greek
g. Cyrillic
h. Armenian
i. Latin Extended Additional
j. Greek Extended

This may be disabled by use of the –nouniupper compiler switch.

4. Graphical representation of the characters is the responsibility of a combination of the Java implementation and operating system. Different graphical representations may be available for displaying upon SYSOUT rather than CONSOLE. Java 2 is generally more capable of displaying and accepting Unicode characters than JDK 1.1.

Alphabets

Alphabets in COBOL are named specifications of coded character sets and collating sequences. The SPECIAL-NAMES paragraph provides the means for naming alphabets and for specifying user-defined coded character sets and collating sequences. A coded character set or collating sequence can be used by specifying its alphabet-name in COBOL statements or entries that reference a coded character set or collating sequence as an operand.

Lexical elements

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

Character-strings

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

COBOL words

A COBOL word is a character-string of not more than 31 characters that form a compiler-directive word, a context-sensitive word, an intrinsic-function-name, a

reserved word, a system-name, or a user-defined word. Each character of a COBOL word that is not a special character word shall be selected from the set of basic letters, basic digits, extended letters, and the basic special characters hyphen and underscore. The hyphen or underscore shall not appear as the first or last character in such words.

Within a source program the following apply:

1. For all COBOL words excluding the words INTEGER, LENGTH, RANDOM, and SUM:
    a. Reserved words form disjoint sets with context-sensitive words, intrinsic-function-names, system-names, and user-defined words.
    b. Compiler-directive words, context-sensitive words, intrinsic-function-names, system-names, and user-defined words form intersecting sets. The same COBOL word may be used as a compiler-directive word, as a context-sensitive word, as an intrinsic-function-name, as a system-name, and as a user-defined word. The classification of a specific occurrence of such COBOL words is determined by the context of the statement, clause, or phrase in which it occurs.
2. For the COBOL words INTEGER, LENGTH, RANDOM, and SUM:
    a. The reserved words INTEGER, LENGTH, RANDOM, and SUM form an identical set with the intrinsic-function-names INTEGER, LENGTH, RANDOM, and SUM. The same COBOL word LENGTH, RANDOM, or SUM may be used as an intrinsic-function-name and as a reserved word. The classification of a specific occurrence of such COBOL words is determined by the context in which it occurs.
    b. The COBOL words INTEGER, LENGTH, RANDOM, and SUM form disjoint sets with user-defined words and system-names. The COBOL words INTEGER, LENGTH, RANDOM, and SUM may not be used as a system-name or as a user-defined word regardless of context.

User-defined words

A user-defined word is a COBOL word that shall be supplied by the user to satisfy the format of a clause or statement.

Types of user-defined words

alphabet-name	paragraph-name
cd-name (obsolete element)	parameter-name
class-name (for object orientation)	program-name
class-name (for truth value proposition)	program-prototype-name
condition-name	property-name
constant-name	record-key-name
data-name	record-name
file-name	report-name
function-prototype-name	screen-name
index-name	section-name
interface-name	symbolic-character
level-number	type-name
method-name	user-function-name
mnemonic-name	

Within a given source element the user-defined words are grouped into the following disjoint sets:

- alphabet-names
- class-names or object-class-names (for object orientation)
- class-names or value-class-names (for truth value proposition)
- condition-names, data-names, property-names, record-key-names, and record-names
- constant-names
- file-names
- index-names
- interface-names
- mnemonic-names
- paragraph-names
- parameter-names
- program-names
- program-prototype-names
- screen-names
- section-names
- symbolic-characters
- user-function-names

Each user-defined word, except level-number, belongs to one of these disjoint sets. Further, all user-defined words within a given disjoint set shall be unique, except as specified in Uniqueness of reference.

With the exception of section-names, paragraph-names, and level-numbers, each user-defined word shall contain at least one basic letter or extended letter. Level-numbers need not be unique; a given specification of a level-number may be identical to any other level-number.

The following user-defined words shall be externalized to the operating environment:

1. program-names of separately-compiled programs, class-names, function-prototype-names, interface-names, method-names, program-prototype-names, and user-function-names
2. data-names, file-names, and record-names of items described with the EXTERNAL attribute.

When specified as a literal, externalized names may contain any characters other than the null character.

The AS phrase specifies a literal to be used as the name that is externalized to the operating environment

When two or more source elements identify something with the same externalized name, they refer to the same instance. No name that is externalized to the operating environment can identify more than one kind of instance.

Alphabet-name

An alphabet-name identifies a specific collating set and character sequence. This relationship is established in the SPECIAL-NAMES paragraph.

Class-name (for object orientation)

A class-name identifies a class, the entity that defines common behavior and implementation for zero, one, or more objects. This relationship is established in the REPOSITORY paragraph or in the CLASS-ID paragraph.

Class-name (for truth value proposition)

A class-name identifies a proposition, for which a truth value can be determined, that the content of a data item consists exclusively of those characters listed in the definition of the class-name. This relationship is established in the SPECIAL-NAMES paragraph.

Condition-name

A condition-name identifies a value, set of values, or range of values defined in the data division, or identifies an on or off status defined in the SPECIAL-NAMES paragraph.

Constant-name

A constant-name identifies a constant declared as a level 78 data item.

Data-name

A data-name identifies a data item described in a data description entry or a record described in a record description entry.

File-name

A file-name identifies a file connector described in a file description entry or a sort-merge file description entry within the file section of the data division.

Index-name

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

Interface-name

An interface-name identifies an interface, a grouping of method prototypes. This relationship is established in the REPOSITORY paragraph or in the CLASS-ID paragraph.

Level-number

A level-number, expressed as a one-digit or two-digit number, indicates the hierarchical position of a data item or the special properties of a data description entry.

Mnemonic-name

A mnemonic-name identifies an implementor-named device-name, feature-name, or switch-name. This relationship is established in the SPECIAL-NAMES paragraph.

Mnemonic devices recognized

Mnemonic Name	*Synonyms*	*Function*
CONSOLE (default)	CRT	Graphical CONSOLE or Text Terminal CONSOLE
SYSOUT, SYSIN	SYSIPT, SYSLIST, SYSLST, SYSOUT-FLUSH, SYSPCH, SYSPUNCH,	Standard output/input stream.
SYSERR		Standard error stream.
SERVLETOUT, SERVLETIN	SERVLET-OUT, SERVLET-IN, SESSION-OUT, SESSION-OUT, SESSIONOUT, SESSIONIN	Servlet or CGI output / input.
PRINTER		Printer virtual device.
C01		Not implemented.
C02		Not implemented.
C03		Not implemented.
C04		Not implemented.
C05		Not implemented.
C06		Not implemented.
C07		Not implemented.
C08		Not implemented.
C09		Not implemented.
C10		Not implemented.
C11		Not implemented.
C12		Not implemented.
S01		Not implemented.
S02		Not implemented.
S03		Not implemented.
S04		Not implemented.
S05		Not implemented.
CSP		Not implemented.
ARGUMENT-NUMBER		ACCEPT to read,
ARGUMENT-VALUE		DISPLAY to set, ACCEPT to read
ENVIRONMENT-NAME		DISPLAY to set,
ENVIRONMENT-VALUE		DISPLAY to set, ACCEPT to read

Paragraph-name

A paragraph-name identifies a paragraph in the procedure division. Paragraph-names are equivalent if they are composed of the same sequence of the same number of COBOL characters.

Parameter-name

A parameter-name identifies a formal parameter of a parameterized class or a parameterized interface.

Program-name

A program-name identifies a program. For a COBOL program, program-name is the name specified in the PROGRAM-ID paragraph of the program's identification division. This program-name is externalized as a lowercase name with hyphens turned to underscores if specified directly; if specified as a literal, the name is externalized directly as given. The name should consist of only upper- and lower-

case letters, digits, and underscores. Other names may work in certain Java implementations, but their use should be avoided for portability. CALL's to upper-case names will map first to the name given, but will also find programs with lower-case, converted names.

Program-prototype-name

A program-prototype-name identifies a program prototype.

Property-name

A property-name identifies a means of getting information out of and passing information back into an object.

Record-key-name

A record-key-name identifies a key associated with an indexed file.

Record-name

A record-name identifies a record described in a record description entry. A record-name may be specified where a data-name is allowed unless specific rules for the format disallow it.

Screen-name

A screen-name identifies a screen description entry in the screen section.

Section-name

A section-name identifies a section in the procedure division.

Symbolic-character

A symbolic-character is a user-defined figurative constant that represents a value specified in the SPECIAL-NAMES paragraph.

User-function-name

A user-function-name identifies a function.

System-names

A system-name is used to communicate with the operating environment. System names are formed in the same manner as all identifiers.

The types of system-names are:

- call-convention-name
- code-name
- computer-name
- device-name
- feature-name
- library-name
- locale-name

- report-attribute-name
- Switch-name
- text-name

Within a given implementation these types of system-names form disjoint sets; a given system-name may belong to one and only one of them.

Call-convention-name

A call-convention-name identifies attributes of the linkage mechanism for a function, method, or program, such as the mechanism for passing arguments, stack management, and name case sensitivity.

Code-name

A code-name identifies a character code set and a collating sequence.

Computer-name

A computer-name may identify the computer upon which the program is to be compiled or run.

Device-name

A device-name identifies an input-output device.

Feature-name

A feature-name identifies a feature of an input-output device.

Library-name

A library-name identifies a COPY library.

Switch-name

A switch-name is identified by:

SWITCH-1 … SWITCH-26
SYSTEM-SWITCH-1 … SYSTEM-SWITCH-26
SWITCH {integer-1-to-26}
SWITCH {nonnumeric-single-character-literal-A-to-Z}

Text-name

A text-name identifies a library text.

Reserved words

A reserved word is one of a specified list of words that is be used in COBOL source program, but that shall not appear in the program as a user-defined word or a system-name. Reserved words shall only be used as specified in the general formats. (See 8.9, Reserved words.)

Reserved words satisfy the following conditions:

1. Reserved words do not begin with the characters '0', ... , '9', 'X', 'Y', or 'Z' except for the reserved words ZERO, ZEROES, and ZEROS.

Additional reserved words are ZERO-FILL, YEAR, YYYYMMDD, and YYYYDDD.

2. Reserved words do not contain only one alphabetic character.
3. Reserved words do not start with 1 or 2 characters followed by '-' except for the reserved words I-O, I-O-CONTROL, and reserved words that begin with 'B-'.
4. Reserved words do not contain two or more contiguous hyphens.

There are three types of reserved words:

- required words
- optional words
- special purpose words

5. Reserved words, with the exception of LC_CURRENCY and locale-category names, do not contain underscores.

Required words

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

Required words are of two types:

1. Key words. Within each format, such words are uppercase and underlined.
2. Special character words. These are the arithmetic operators, the concatenation operator, the invocation operator, and the relation characters.

Optional words

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

Special purpose words

The types of special purpose words are:

- figurative constants
- predefined address
- predefined object identifiers
- special registers

Figurative constants

Certain reserved words are used to name and reference specific constant values. These reserved words are specified in figurative constant values.

Figurative constants may also be defined in the FIGURATIVE CONSTANTS paragraph in the CONFIGURATION SECTION.

Predefined address

The reserved word NULL is used as a predefined address.

Predefined object identifiers

Reserved words used as predefined object identifiers are:

- NULL
- SELF
- SUPER

Further specification of predefined object identifiers is given in Identifiers.

Special registers

The following reserved words are used to name and reference special registers:

- CURRENT-DATE
- CURRENT-TIME
- FAC
- LENGTH
- LINAGE-COUNTER
- LINE-COUNTER
- PAGE-COUNTER
- RECORD POSITION
- SIZE
- TIME-OF-DAY
- WHEN-COMPILED

Further specification of special registers is given in Special registers.

Context-sensitive words

A context-sensitive word is a COBOL word that is reserved only in the general formats in which it is specified. The same word may also be used as an intrinsic-function-name, a user-defined word, or a system-name.

Intrinsic-function-names

An intrinsic-function-name is a COBOL word that identifies a specific intrinsic function (see 15.5, Summary of functions). With the exception of the words INTEGER, LENGTH, RANDOM, and SUM, a word that is an intrinsic-function-name may appear in a different context in a program as a context-sensitive word, a system-name, or a user-defined word.

Literals

A literal is a character string representing a data value derived from the ordered set of characters of which the literal is composed or is defined by a reserved word that references a figurative constant. Each literal possesses a class and category: alphanumeric, national, or numeric.

The paired quotation symbols specified in the opening and closing delimiters of alphanumeric literals may be either apostrophes or quotation marks. Both forms may be used within a single source unit.

Hexadecimal digits are used to specify the value of the literal in the hexadecimal-alphanumeric and hexadecimal-G formats of literals. The hexadecimal digits are the basic digits '0' through '9' and the basic letters 'A' through 'F'. When used as hexadecimal digits, the lowercase letters 'a' through 'f' are equivalent to the corresponding uppercase letters 'A' through 'F'.

Alphanumeric literals

Alphanumeric literals are of the class and category alphanumeric.

General format

Format 1 (alphanumeric):
{ " {character-1} …" | ' {character-1} … ' }

Format 2 (mixed-text-alphanumeric):
{ T" {character-1} … " | T' {character-1} … ' }

Format 3 (hexadecimal-alphanumeric):
{ X" {hex-character-sequence-1} …" | X' {hex-character-sequence-1} …' }

Format 4 (octal-alphanumeric)
{ O" {octal-character-sequence-1} …" | O' {octal-character-sequence-1} …' }

Format 5 (octal-alphanumeric-2 (HP-style))
{ %{octal-character-sequence-1} … }

Format 6 (binary-alphanumeric)
{ I" {binary-character-sequence-1} …" | I' {binary-character-sequence-1} …' }

Format 7 (decimal-alphanumeric)
{ D" {decimal-character-sequence-1} …" | D' {decimal-character-sequence-1} …' }

Syntax rules

ALL FORMATS

The length of an alphanumeric literal, excluding the separators that delimit the literal, shall be greater than zero and less than or equal to 160 alphanumeric character positions.

This limit is raised to 2047.

Elastic COBOL currently cannot embed null characters (0) in alphanumeric literals.

FORMATS 1 AND 2

1. Character-1 may be any character in the computer's character set.
2. If character-1 is to represent the quotation symbol used in the opening delimiter, two contiguous matching quotation symbol characters shall be specified to represent a single occurrence of that character.
3. The two contiguous quotation symbols used to represent a single quotation symbol character shall be in the same coded character set representation.
4. If character-1 is a COBOL basic letter, the lowercase and uppercase letters are distinct.

FORMAT 1

Do not include national characters in a format 1 literal.

FORMAT 2

National characters in external media format shall be permitted in a format 2 alphanumeric literal.

FORMAT 3

1. Hex-character-sequence-1 shall be composed of hexadecimal digits.
2. Each hex-character-sequence-1 shall consist of two (2) characters.
3. The value of each hex-character-sequence-1, as specified in general rule 6, shall be less than 256.

FORMAT 4

Octal-character-sequence-1 is a three (3) character sequence of octal digits from 000-377.

FORMAT 5

Octal-character-sequence-1 is a three (3) character sequence of octal digits from 000-377. This format is for HP compatibility only; do not use in new code.

FORMAT 6

Binary-character-sequence-1 is an eight (8) character sequence of binary digits 0 or 1.

FORMAT 7

Decimal-character-sequence-1 is a three (3) character sequence of decimal digits from 000-255.

General rules

ALL FORMATS

1. The separators that delimit the alphanumeric literal shall not be included in the value of the alphanumeric literal.
2. Alphanumeric literals are of the class and category alphanumeric.
3. The value of an alphanumeric literal at run time shall be the value represented by character-1.
4. When national characters in external media format appear in the value of the literal, control functions, if any, essential to the representation of the external media format of character-1 shall be included in the value of the literal.
5. When national characters in external media format appear in the value of the literal, the value of the literal shall be treated at execution time as a string of alphanumeric characters except for purposes of explicit conversion between classes as defined for the DISPLAY-OF and NATIONAL-OF functions.

FORMAT 3

The value of the literal at run time is the series of native characters specified by the series of hex-character-sequence-1. For each hex-character-sequence-1, the value resulting from the following expression is the ordinal position of the character in the value of that native coded character set:

$$1 + \sum_{1}^{\#\ digits} \text{digit-value} * (16 ** (\text{position-from-right} - 1))$$

- where digit-value for a numeric digit is the value that digit has when specified as a numeric literal and digit-value for 'A' is 10, 'B' is 11, 'C' is 12, 'D' is 13, 'E' is 14, and 'F' is 15.
- and where position-from-right is such that the rightmost digit is 1, the digit to the left of it is 2, etc.

Numeric literals

Numeric literals are of the class and category numeric.

Fixed-point numeric literals

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

1. A literal shall contain at least one digit.
2. A literal shall not contain more than one sign character. If a sign is used, it shall appear as the leftmost character of the literal. If the literal is unsigned, the literal is nonnegative.
3. A literal shall not contain more than one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, the literal is an integer.
4. The value of a fixed-point numeric literal is the algebraic quantity represented by the characters in the fixed-point numeric literal. The size of a fixed-point numeric literal in standard data format characters is equal to the number of digits in the string of characters as specified by the user.

Floating-point numeric literals

The rules for the formation and value of floating-point numeric literals are:

1. A floating-point numeric literal is formed from two fixed-point numeric literals separated by the letter 'E' without any spaces.
2. The literal to the left of the 'E' represents the significant. It may be signed and shall include a decimal point. The significant shall be 1 through 31 digits in length. If the significant is signed, the floating-point numeric literal is considered to be signed. If the significant is unsigned, the floating-point numeric literal is considered to be positive.
3. The literal to the right of the 'E' represents the exponent. It may be signed and shall have a maximum of three digits and no decimal point. The maximum permitted value of the exponent is implementor-defined.
4. If all the digits in the significant are zero, then all the digits of the exponent shall also be zero and neither significant nor exponent shall have a negative sign.

5. The value of a floating-point numeric literal is the algebraic product of the value of its significant and the quantity derived by raising ten to the power of the exponent.

General format

Format 1 (decimal)

{digit} …

Format 2 (hexadecimal-numeric):

{ H" {hex-character-sequence-1} …" | H' {hex-character-sequence-1}…'

| H#{hex-character-sequence-1} | X#{hex-character-sequence-1}}

Format 3 (octal-numeric)

{ Q" {octal-character-sequence-1} …" | Q' {octal-character-sequence-1} …'

|Q#{octal-charater-sequence-1} | O#{octal-character-sequence-1}}

Format 4 (binary-numeric)

{ B" {binary-character-sequence-1} …" | B' {binary-character-sequence-1} …'

|B#{binary-character-sequence-1} | I#{binary-character-sequence-1}}

Format 5 (decimal-numeric)

{ C" {decimal-character-sequence-1} …" | C' {decimal-character-sequence-1} …'

|C#{decimal-character-sequence-1} | D#{decimal-character-sequence-1}}

General rules

ALL FORMATS

The result is a numeric literal value.

FORMAT 1

This is the standard numeric format.

FORMAT 2

1. Hex-character-sequence-1 shall be composed of hexadecimal digits.
2. Each hex-character-sequence-1 shall consist of two (2) characters.
3. The value of each hex-character-sequence-1, as specified in general rule 6, shall be less than 256.
4. AcuCOBOL compatibility H is treated as alphanumeric rather than numeric; if the second character is not a '#'

FORMAT 3

Octal-character-sequence-1 is a three (3) character sequence of octal digits from 000-377.

FORMAT 4

Binary-character-sequence-1 is an eight (8) character sequence of binary digits 0 or 1.

FORMAT 5

Decimal-character-sequence-1 is a three (3) character sequence of decimal digits from 000-255.

National literals

National literals are of the class and category national.

General format

Format 1 (national)

{ N”{character-1} …” | N’{character-1}…’ }

Format 2 (hexadecimal-national (IBM compatibility))

{ G”{hex-character-sequence-1}…” | G’{hex-character-sequence-1}…’ }

Syntax rules

ALL FORMATS

If character-1 is to represent the quotation symbol used in the opening delimiter, two contiguous matching quotation symbol characters shall be specified to represent a single occurrence of that character.

FORMAT 1

1. Character-1 shall be any character in the computer's national character set.
2. The length of a national literal, excluding the separators that delimit the literal, shall be greater than zero and less than or equal to 160 national character positions.

FORMAT 2

1. Hex-character-sequence-1 shall be composed of hexadecimal digits.
2. Each hex-character-sequence-1 shall consist of 4 hexadecimal digits.
3. The value of each hex-character-sequence-1, as specified in general rule 4, shall be less than the value that the implementor has specified as the maximum value of the hexadecimal digits that map to a national character.
4. The < character is the literal SHIFT-OUT character.
5. The > character is the literal SHIFT-IN character.
6. This form is for IBM compatibility only; do not use this form in new code.

General rules

ALL FORMATS

1. The separators that delimit the national literal shall not be included in the value of the national literal.
2. National literals are of the class and category national.

FORMAT 1

The value of a national literal at run time shall be the value represented by character-1.

FORMAT 2

The value of the literal at run time is the series of national characters specified by the series of hex-character-sequence-1. For each hex-character-sequence-1, the value resulting from the following expression is the ordinal position of the character in the national coded character set:

- where digit-value for a numeric digit is the value that digit has when specified as a numeric literal and digit-value for 'A' is 10, 'B' is 11, 'C' is 12, 'D' is 13, 'E' is 14, and 'F' is 15, and where position-from-right is such that the rightmost digit is 1, the digit to the left of it is 2, etc.

Figurative constant values

Figurative constant values are generated by the compiler and referenced through the use of the reserved words given below. When used as figurative constants, these words shall not be bounded by the opening and closing delimiters of literals. The singular and plural forms of figurative constants are equivalent and may be used interchangeably.

The figurative constant value and the reserved words used to reference them are as follows:

1. [ALL] ZERO, [ALL] ZEROS, [ALL] ZEROES: Represents the numeric value '0',, or one or more of the character '0' from the computer's character set, depending on context.
2. [ALL] SPACE, [ALL] SPACES: Represents one or more of the character space from the computer's character set.
3. [ALL] HIGH-VALUE, [ALL] HIGH-VALUES: Except in the SPECIAL-NAMES paragraph, represents one or more of the character that has the highest ordinal position in the program collating sequence.
4. [ALL] LOW-VALUE, [ALL] LOW-VALUES: Except in the SPECIAL-NAMES paragraph, represents one or more of the character that has the lowest ordinal position in the program collating sequence.
5. [ALL] QUOTE, [ALL] QUOTES: Represents one or more of the character "" in the computer's character set. The word QUOTE or QUOTES may not be used in place of a quotation symbol in a source program to bind a literal. With the APOST directive, the character represented is (‘).
6. ALL literal: Represents all or part of the string generated by successive concatenations of the characters comprising the literal. The literal shall be an alphanumeric, Boolean, or national literal, any of which may be a concatenation expression. The literal shall not be a figurative constant.
7. [ALL] symbolic-character: Represents one or more of the character specified as the value of this symbolic-character in the FIGURATIVE CONSTANTS clause of the CONFIGURATION SECTION.

When a figurative constant is used in a context requiring national characters, the figurative constant shall represent a national character value. Otherwise, when a figurative constant represents a character value, the figurative constant represents an alphanumeric character value. In both cases, the character value representation of the figurative constant ZERO (ZEROS, ZEROES), SPACE (SPACES), and QUOTE (QUOTES) shall be the value of the character '0', space, and "", respectively, in the computer's character set.

When a figurative constant represents a string of one or more characters, the length of the string is determined from context by applying the following rules in order:

1. When a figurative constant is specified in a VALUE clause, or when a figurative constant is associated with another data item, the string of characters specified by the figurative constant is repeated character by character on the right until the size of the resultant string is greater than or equal to the number of character positions in the associated data item. This resultant string is then truncated from the right until the number of character positions remaining is equal either to 1 or to the number of character positions in the associated data item, whichever is greater. This is done prior to and independent of the application of any JUSTIFIED clause that may be associated with the data item.

   NOTE - A figurative constant is associated with a data item when, for example the figurative constant is moved to or compared with that data item.

2. When a figurative constant, other than ALL literal, is not associated with a VALUE clause or another data item, the length of the string is one character.

   NOTE - For example, when the figurative constant appears in a DISPLAY, STOP, STRING, or UNSTRING statement, it is one character.

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

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

1. When a figurative constant is specified in a concatenation expression, the length of the string is one character.
2. If the literal is restricted to a numeric literal, the only figurative constant permitted is ZERO (ZEROS, ZEROES) without the word ALL.
3. The figurative constant ALL literal, when the length of the literal is greater than one, is not permitted to be associated with a numeric or numeric-edited item.
4. When a figurative constant other than ALL literal is used, the word ALL is redundant and is used for readability only.
5. A figurative constant shall not be specified where an alphanumeric literal is used to identify a method.

Except in the SPECIAL-NAMES paragraph, when the figurative constants HIGH-VALUE(S) or LOW-VALUE(S) are used in the source program, the actual characters associated with each figurative constant depend upon the program collating sequence specified. When LOW-VALUE and HIGH-VALUE are used in a context requiring national characters, the character represented shall be from the national program collating sequence; otherwise, the character represented shall be from the alphanumeric program collating sequence.

Punctuation characters

Punctuation characters are characters used in the formation of separators.

Punctuation characters

Character	Meaning
,	comma
;	semicolon

Character	Meaning
:	colon
.	period (full stop)
"	quotation mark
'	apostrophe
(	left parenthesis
)	right parenthesis
	space
=	equal sign

Separators

A separator is one, two, or three contiguous characters formed according to the following rules:

1. The punctuation character space is a separator. Anywhere a space is used as a separator or as part of a separator, more than one space may be used. All spaces immediately following the separators comma, semicolon, or period are considered part of that separator and are not considered to be the separator space.
2. Except when the comma is used in a PICTURE character-string, the punctuation characters comma and semicolon, immediately followed by a space, are separators that may be used anywhere the separator space is used. They may be used to improve program readability.
3. The punctuation character period, when followed by a space is a separator. It shall be used only to indicate the end of a sentence, or as shown in formats.
4. The punctuation characters right and left parentheses are separators. Except in pseudo-text, parentheses may appear only in balanced pairs of left and right parentheses delimiting subscripts, a list of function arguments, reference modifiers, arithmetic expressions, or conditions.
5. The opening delimiters and closing delimiters of literals are separators. Either an apostrophe or a quotation mark may be used as the quotation symbol character in opening and closing delimiters.

   The opening delimiters of literals are

   - a quotation symbol
   - the two contiguous characters N", X', etc. used to form literals

   The closing delimiters of literals are:

   - a quotation mark when the opening delimiter uses a quotation mark
   - an apostrophe when the opening delimiter uses an apostrophe

   The opening delimiter shall be immediately preceded by a space, left parentheses, or opening pseudo-text delimiter. The closing delimiter shall be immediately followed by one of the separator space, comma, semicolon, period, right parenthesis, or closing pseudo-text delimiter. Separators immediately preceding the opening delimiter are not part of the opening delimiter. Separators immediately following the closing delimiter are not part of the closing delimiter.

6. Pseudo-text delimiters are separators. An opening pseudo-text delimiter shall be immediately preceded by a space; a closing pseudo-text delimiter shall be immediately followed by one of the separator space, comma, semicolon, or period. Pseudo-text delimiters may appear only in balanced pairs delimiting pseudo-text.
7. The punctuation character colon is a separator and is required when shown in the general formats.
8. The separator space may optionally immediately precede all separators except:
   a. As specified by reference format rules (see Reference format.)
   b. The closing delimiter of a literal.
   c. The opening pseudo-text delimiter, where the preceding space is required.
9. The separator space may optionally immediately follow any separator except the opening delimiter of a literal. A space following the opening delimiter of a literal shall be part of the literal and not a separator.

Any punctuation character appearing as part of the specification of a PICTURE character-string or numeric literal is not considered as a punctuation character, but rather as a symbol used in the specification of that PICTURE character-string or numeric literal. PICTURE character-strings are delimited only by the separator space, comma, semicolon, or period.

The rules established for the formation of separators do not apply to the content of alphanumeric, or national literals or comments.

References

References identify elements referred to during compilation of source unit or execution of a run unit. The reserved words and types of names specified in Lexical elements are forms of reference. Additional forms of reference are condition-names and identifiers.

Condition-name

A condition-name identifies a specific value, set of values, or range of values, within a complete set of values that a data item may assume. The data item itself is called a conditional variable.

Condition-names may be defined in the data division or in the SPECIAL-NAMES paragraph within the environment division where a condition-name shall be assigned to the on status or off status, or both, of the switches SWITCH-1 through SWITCH-26.

A condition-name is used in conditions as an abbreviation for the relation condition; this relation condition posits that the associated conditional variable is equal to one of the set of values to which that condition-name is assigned. A condition-name is also used in a SET statement, indicating either that a value is moved to the associated conditional variable that makes the condition-name either 'true' or 'false', depending on the format of the SET statement, or that SWITCH-1 through SWITCH-26 is set to an 'on' or 'off' status.

General format

Format 1 (switch-status-condition-name):
condition-name-1

Format 2 (qualified-condition-name-with-subscripts):
condition-name-2 [{IN|OF} data-name-2] …
[{IN|OF} {file-name-1}]
[({subscript-1} …)]

Syntax rules

FORMAT 1

Condition-name-1 shall be associated with a switch-name in the SPECIAL-NAMES paragraph.

FORMAT 2

If the conditional variable associated with condition-name-2 requires subscripting, condition-name-2 shall be subscripted with the same number of subscripts required for the conditional variable.

Identifiers

Identifier

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

General format

Format 1 (function identifier):
function-identifier-1

Format 2 (qualified-data-name-with-subscripts):
data-name-1 [{IN|OF} data-name-2] …
[{IN|OF} {file-name-1}]
[({subscript-1} …)]

Format 3 (reference-modification):
identifier-1 reference-modifier-1

Format 6 (predefined-object):
NULL
SELF
SUPER

Format 9 (predefined-address)
NULL

Format 10 (address-identifier)
data-address-identifier-1

Syntax rules

ALL FORMATS

Identifier is defined recursively: whenever the format for an identifier allows another identifier to be specified, that other identifier may be any of the formats for an identifier, including the one being defined provided the rules for each format are followed.

FORMAT 1

Function-identifier-1 is defined by Function Identifier.

FORMAT 2

1. The words IN and OF are equivalent.
2. If data-name-1 is not unique in the scope of names of the current source element, then it shall be followed by a combination of qualifiers and subscripts necessary for uniqueness of reference in accordance with Scope of names.
3. Subscripts are defined by Subscripts.

FORMAT 3

Reference-modifier-1 is defined by Reference-modifier.

FORMAT 4

Predefined-object identifiers are defined by NULL; and SELF and SUPER.

FORMAT 5

Predefined-address NULL is defined in Predefined-address.

FORMAT 6

Address-identifiers are defined by Data-address-identifier.

General rules

The order in which the various components of an identifier are applied is as follows, with the first to be applied listed first:

1. predefined-object identifiers are considered elementary identifiers
2. OF or IN for data-name qualification, qualifies the (possibly already qualified) data-name on the left with the cd-name, file-name, report-name or data-name on the right
3. a subscript applies to the fully qualified data-name on the left
4. the object-modifier applies to the object identifier on the left
5. OF for object properties applies the property name on the left to the object identifier on the right
6. reference modification applies to the identifier on the left.

Function-identifier

A function-identifier references the unique data item that results from the evaluation of a function.

General format

Syntax rules

1. A function-identifier shall not be specified as a receiving operand.
2. If a function may optionally have zero arguments, a left parenthesis immediately following function-name-1 shall be the left delimiter of that function's argument list.

NOTE - Putting such a function in parentheses separates the function from the next item starting with a left parenthesis in the list of arguments and subscripts. For example:

```
FUNCTION MAX ((FUNCTION RANDOM) (A)  B)
```

3. Argument-1 shall be an identifier, a literal, or an arithmetic expression. Specific rules governing the number, class, and category of argument-1 are given for intrinsic functions in the definition of that intrinsic function in Intrinsic functions, and for user-defined functions in Conformance for parameters and returning items.
4. A numeric function shall not be specified where an integer operand is required, even though a particular reference of the numeric function might yield an integer value.
5. An integer function other than the integer form of the ABS function shall not be specified where an unsigned integer is required.
6. If function-prototype-name-1 is specified, the rules for conformance specified in Conformance for parameters and returning items, shall apply.

General rules

1. At the time reference is made to a function, its arguments are evaluated individually in the order specified in the list of arguments, from left to right. An argument being evaluated may itself be a function-identifier or may be an expression containing function-identifiers. There is no restriction preventing the function referenced in evaluating an argument from being the same function as that for which the argument is specified. Additional rules for intrinsic functions are given in Intrinsic functions, for user-defined functions in General rules of the procedure division and in Conformance for parameters and returning items.
2. A function is a temporary data item whose value is determined when the function is referenced at run time.

   If intrinsic-function-name-1 is specified, the temporary data item is an elementary data item whose description and category are specified by the definition of that intrinsic function in Intrinsic functions.

3. Evaluation of the function-identifier proceeds as follows:
   a. Each argument-1 is evaluated at the beginning of the evaluation of the function-identifier. If an exception condition exists, no function is activated and execution proceeds as specified in general rule 6g. If an exception condition does not exist, the values of argument-1 are made available to the activated function at the time control is transferred to that function.

b. The runtime system attempts to locate the function being activated. If function-prototype-name-1 is specified, the rules are specified in Scope of names and Conventions for function-prototype-names and program-prototype-names.

Reference-modifier

Reference modification defines a unique data item by specifying an identifier, a leftmost position, and a length.

General format

Syntax rules

1. Identifier-1 shall reference a data item that is one of the following:
   - an elementary item of category alphanumeric or national, Identifier-1 shall not include a type-modifier with the type-name-1 phrase.
2. Leftmost-position and length shall be arithmetic expressions.
3. Unless otherwise specified, reference modification is allowed anywhere an identifier referencing a data item of class alphanumeric or national is permitted.

General rules

1. Leftmost-position shall represent a Boolean position, alphanumeric position, or national position when identifier-1 references a Boolean, alphanumeric, or national data item, respectively.
2. If the data item referenced by identifier-1 is explicitly or implicitly described as usage DISPLAY and its category is other than alphanumeric, it shall be operated upon for purposes of reference modification as if it were redefined as a data item of class and category alphanumeric of the same size as the data item referenced by identifier-1.
3. If the data item referenced by identifier-1 is explicitly or implicitly described as usage NATIONAL and its category is other than national, it shall be operated upon for purposes of reference modification as if it were redefined as a data item of class and category national of the same size as the data item referenced by identifier-1.
4. Each position of a data item referenced by identifier-1 is assigned an ordinal number incrementing by one from the leftmost position to the rightmost position. The leftmost position is assigned the ordinal number one. If the data description entry for identifier-1 contains a SIGN IS SEPARATE clause, the sign position is assigned an ordinal number within that data item.
5. Reference modification creates a unique data item that is a subset of the data item referenced by identifier-1. This unique data item is defined as follows:
6. The unique data item is considered to be an elementary data item without the JUSTIFIED clause. The unique data item has the same class, category, and usage as that defined for identifier-1, except that the categories numeric,

numeric-edited, and alphanumeric-edited are considered class and category alphanumeric.

Subscripts

Subscripts are used when reference is made to an individual element within a table of like elements.

General format

NOTE - Condition-name-1 and data-name-1 are shown for context and are not part of the subscript general format.

Syntax rules

1. The data description entry containing data-name-1 or the data-name associated with condition-name-1 shall contain an OCCURS clause or shall be subordinate to a data description entry that contains an OCCURS clause.
2. Except as defined in syntax rule 4, when a reference is made to a table element, the number of subscripts shall equal the number of OCCURS clauses in the description of the table element being referenced. This allows a maximum of seven subscripts to be specified. When more than one subscript is required, the subscripts are written in the order of successively less inclusive dimensions of the table.
3. Index-name-1 shall correspond to a data description entry in the hierarchy of the table being referenced that contains an INDEXED BY phrase specifying that index-name.
4. Each table element reference shall be subscripted except when such reference appears:
   a. As the subject of a SEARCH statement.
   b. In a REDEFINES clause.
   c. In the KEY IS phrase of an OCCURS clause.
   d. In a SORT statement that references a table.
   e. In the FROM, TO, or USING clauses of a screen description entry when the subject of the entry has an OCCURS clause.
5. The subscript ALL may be used only when the subscripted identifier is used as a function argument and may not be used when condition-name-1 is specified. (See Arguments.)

General rules

1. A subscript is determined as follows:
   a. If ALL is specified, the subscript is all of the possible values of a subscript for the associated table as specified in the rules for the statements for which ALL is allowed.

b. If index-name-1 is specified, the subscript is the occurrence number represented by the value of the index referenced by index-name-1 modified by integer-1. The mapping of the value of the index referenced by index-name-1 to an occurrence number is defined by the implementor. If integer-1 is specified, the subscript is the occurrence number derived from the index incremented by the value of integer-1 (when the operator + is used) or decremented by the value of integer-1 (when the operator B is used).

2. The value of a subscript shall be a positive integer. The lowest possible occurrence number represented by a subscript is 1, which identifies the first element of any given dimension of a table. Each successive element within that dimension of the table is referenced by occurrence numbers of 2, 3, ... The highest permissible occurrence number for any given dimension of the table is the maximum number of occurrences of the item as specified in the associated OCCURS clause. If the value of the subscript is less than one or greater than the highest permissible occurrence number, a runtime exception is given.

NULL

NULL is a predefined object identifier that references the NULL object.

General format

NULL

Syntax rules

NULL shall not be specified as a receiving operand.

General rules

NULL always references the same object, the NULL object.

SELF and SUPER

SELF and SUPER are predefined object identifiers that reference the object on which the current method is executing.

General format

{ SELF | SUPER }

Syntax rules

1. This identifier format shall not be specified as a receiving operand.
2. SUPER may be specified only as the object in an object-property identifier or as the object used to invoke a method with the invoke statement or an in-line invocation of a method.

General rules

1. The predefined object identifiers SELF or SUPER both references the object that was used to invoke the method in which SELF or SUPER appear.
2. If SELF is specified for a method invocation, the method resolution is based upon the set of methods defined for the runtime class of the object referenced by SELF.

   NOTE - The method resolution is not limited to the methods that are defined for the class that contains the method invocation. The object referenced by SELF at run time may be an object of a subclass of the class that contains the invocation. Thus method invocation through the predefined object identifier SELF uses the same method binding mechanism as is used for any other object identifier, based on the runtime class of the object..
3. If SUPER is specified for a method invocation, the method resolution shall ignore all the methods defined in the class containing the invocation and all the methods defined in any subclass of that class. Thus the invoked method will be one that is inherited from a superclass.
4. If class-name-1 is specified, the search for the method shall include only those methods defined for class-name-1.

Predefined-address

NULL is a predefined address of class pointer.

General format

NULL

Syntax rules

This format may be used only as a sending operand in a SET statement; as an argument in a program-prototype format program call, a function-prototype format function call, or a method invocation; or in a data-pointer or program-pointer relation-condition.

General rules

1. When associated with a data-pointer, the predefined address NULL is of category data-pointer and always references the same address, the NULL address, and is guaranteed not to represent the address of any data item.
2. When associated with a program-pointer, the predefined address NULL is of category program-pointer and always references the same address, the NULL address, and is guaranteed not to represent the address of any program.

General Format

ADDRESS OF identifier-1

Syntax rules

1. Identifier-1 shall reference a data item defined in the file section, working-storage section, local-storage section, or linkage section.
2. Identifier-1 shall not reference an object reference.
3. This identifier format shall not be specified as a receiving operand.

General rules

Data-address-identifier creates a unique data item of class pointer and category data-pointer that contains the address of identifier-1.

External switch

An external switch is a software device, SWITCH-1 through SWITCH-26, that is used to indicate that one of two alternate states exists. These alternate states are referred to as the on status and the off status of the associated external switch.

The status of an external switch may be interrogated by testing condition-names associated with that switch. The association of a condition-name with an external switch and the association of a user-specified mnemonic-name with the name SWITCH-1 through SWITCH-26 that names an external switch are established in the SPECIAL-NAMES paragraph of the environment division.

Switches are set according to the main program and are shared among all programs in the current run-unit. The switches are set from the command-line using parameters /A through /Z, /a through /z, or /1 through /9. All switches are initially in the off position.

The status of all switches may be altered by the SET statement.

Uniqueness of reference

Every user-defined name in a source element is assigned, by the user, to name a resource that is to be used in solving a data processing problem. (See User-defined words.) In order to use a resource, a statement in a source element shall contain a reference that uniquely identifies that resource. In order to ensure uniqueness of reference, a user-defined name may be qualified, subscripted, or reference modified as described in the following paragraphs.

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

Every user-defined name explicitly referenced in a compilation group shall be uniquely referenced because either:

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

2. It is unique within the context of a REDEFINES clause.
3. The name exists within, or is associated with a data definition entry within, a hierarchy of names such that reference to the name may be made unique by mentioning one or more of the higher level names in the hierarchy.

   These higher level names are called qualifiers and this process that specifies uniqueness is called qualification. Identical user-defined names may appear in a source unit; however, uniqueness shall then be established through qualification for each user-defined name explicitly referenced, except as specified in rules 2 and 3. All available qualifiers need not be specified so long as uniqueness is established. Reserved words naming the special registers require qualification to provide uniqueness of reference whenever a source unit would result in more than one occurrence of any of these special registers. A paragraph-name or section-name appearing in a source element may not be referenced from any other source element.
4. A source element is contained within a source element or contains another source element. (See Scope of names.)

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

General format

Format 1 (data-or-condition-or-index-name):
{ data-name-1 | condition-name-1 | index-name-1 }
{ {IN|OF} {file-name-1} | { {IN|OF} data-name-2 } … [{IN|OF} file-name-1] } }

Format 2 (procedure-name):
Paragraph-name-1 {IN|OF} section-name-1

Format 3 (library-text-name):
Text-name-1 {IN|OF} library-name-1

Format 4 (linage-counter):
LINAGE-COUNTER {IN|OF} filename-2

Format 5 (screen-name):
Screen-name-1 { {IN|OF} screen-name-2 } …

Format 6 (record-key-name):
Record-key-name-1 [{IN|OF} file-name-3]

Syntax rules

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

2. A name may be qualified even though it does not need qualification; if there is more than one combination of qualifiers that ensures uniqueness, then any such set may be used.
3. The words IN and OF are equivalent.
4. In format 1, each qualifier shall be the name associated with a level indicator, the name of a group item to which the item being qualified is subordinate, or the name of the conditional variable with which the condition-name being qualified is associated. Qualifiers are specified in the order of successively more inclusive levels in the hierarchy. For a condition-name, the hierarchy is that of the associated conditional variable. No more than fifty qualifiers shall be specified.
5. In format 1, data-name-1 or data-name-2 may be a record-name.
6. If explicitly referenced, a paragraph-name shall not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION shall not appear. A paragraph-name need not be qualified when referred to from within the same section. A paragraph-name or section-name appearing in a program may not be referenced from any other source element.

   Paragraph names shall not be duplicated within a section.
7. LINAGE-COUNTER shall be qualified each time it is referenced if more than one file description entry containing a LINAGE clause has been specified in the source unit.

Explicit and implicit references

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

Scope of names

When source units are directly or indirectly contained within other source units, each source unit may use identical user-defined words to name items independent of the use of these user-defined words by other source units. (See User-defined words.) When identically named items exist, a source unit's reference to such a name, even when it is a different type of user-defined word, is to the item which that source unit describes rather than to the item, possessing the same name, described in another source unit.

The following types of user-defined words may be referenced only by statements and entries in that source unit in which the user-defined word is declared:

- paragraph-name
- section-name

The following types of user-defined words may be referenced by any source unit provided that the compiling system supports the associated library or other system and the entities referenced are known to that system:

- library-name
- text-name

The following types of names, when they are declared within a configuration section, may be referenced only by statements and entries either in that source unit that contains a configuration section or in any source unit contained within that source unit:

- alphabet-name
- class-name
- condition-name
- mnemonic-name
- symbolic-character

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

- class-name (for object orientation)
- condition-name
- constant-name
- data-name
- file-name
- index-name
- interface-name
- method-name
- program-name
- record-name
- screen-name

Conventions for program-names

The program-name of a program is declared in the PROGRAM-ID paragraph of the program's identification division. A program-name may be referenced only by the

CALL statement, the CANCEL statement, the SET statement, and the end program marker. The program-names allocated to programs constituting a run unit are not necessarily unique but, when two programs in a run unit are identically named, at least one of those two programs shall be directly or indirectly contained within another separately-compiled program that does not contain the other of those two programs.

The following rules regulate the scope of a program-name for the CALL, CANCEL, and SET statements:

1. If the program-name is that of a program that does not possess the common attribute and that is directly contained within another program, that program-name may be referenced only by statements included in that containing program or the program itself.
2. If the program-name is that of a program that does possess the common attribute and that is directly contained within another program, that program-name may be referenced only by statements included in that containing program and any programs directly or indirectly contained within that containing program, except any programs contained in that program possessing the common attribute.
3. If the program-name is that of a program that is separately compiled, that program-name may be referenced by statements included in any program in the run unit, except programs it directly or indirectly contains.

Conventions for condition-names, data-names, file-names, record-names, report-names, screen-names, and type-names

When condition-names, data-names, file-names, record-names, screen-names, and type-names are declared in a source element, these names may be referenced only by that source element except when one or more of the names is global and the source element contains other source elements.

The requirements governing the uniqueness of the names allocated by a single source element to be condition-names, data-names, file-names, record-names, screen-names, and type-names are explained elsewhere in these specifications. (See User-defined words.)

A source element may not reference any condition-name, data-name, file-name, record-name, screen-names, or type-names declared in any source element it contains.

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

When a source element, source element B, is directly contained within another source element, source element A, and both source elements may define a condition-name, a data-name, a file-name, a record-name, or a screen-name using the same user-defined word. When such a duplicated name is referenced in source element B, the following rules are used to determine the referenced item:

1. The set of names to be used for determination of a referenced item consists of all names that are defined in source element B and all global names that are

defined in source element A and in any source elements that directly or indirectly contain source element 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 items is identified.

2. If only one item is identified, it is the referenced item.
3. If more than one item is identified, no more than one of them may have a name local to source element B. If zero or one of the items has a name local to source element B, the following rules apply:
   a. If the name is declared in source element B, the item in source element B is the referenced item.
   b. Otherwise, if source element A is contained within another source element, the referenced item is:
      - The item in source element A if the name is declared in source element A.
      - The item in the containing source element if the name is not declared in source element A and is declared in the source element containing source element A. This rule is applied to further containing source elements until a single valid name has been found.

Conventions for index-names

If a data item possessing the global attribute includes a table described with an index-name, that index-name 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 may not be qualified.

Conventions for class-names and interface-names

The class-name of a class referenced within a source element shall be either the name of the containing class definition or declared in the REPOSITORY paragraph of that or a containing source element.

Within a compilation group, there shall be at most one class definition for a given class-name.

The interface-name of an interface referenced within a source element shall be either the name of the containing interface definition or declared in the REPOSITORY paragraph of that or a containing source element.

A class-name or interface-name declared in the REPOSITORY paragraph of a source element may be used in that source element and any nested source unit.

Class and category of data

Every elementary data item, every literal, and every function has a class and a category. The class and category of a data item are defined by its picture character string, by the BLANK WHEN ZERO clause, or by its usage; the class and category

of an intrinsic function are specified by the definition of that intrinsic function in Intrinsic functions; the class and category of a user function are specified by the description of the item specified in the RETURNING phrase of the procedure division header of the function prototype; the class and category of a literal are defined in Literals.

When the TYPE clause is specified in the data description of a strongly typed item, the class of the item is the specific type-name specified.

The category of a group item is alphanumeric.

Table 2, Category and class relationships for elementary items, depicts the relationship of categories to classes of data for untyped elementary items.

Category and class relationships for elementary items

Class	Category
Alphabetic	Alphabetic
Alphanumeric	Alphanumeric
Alphanumeric-edited	Numeric-edited
Index	Index
National	National National-edited
Numeric	Numeric
Object	Object-reference
Pointer	Data-pointer Program-pointer

Operators

Arithmetic operators

There are five binary arithmetic operators and two unary arithmetic operators that may be used in arithmetic expressions. They are represented by specific characters that shall be preceded by a space and followed by a space except that no space shall be required between a left parenthesis and a unary operator or between a unary operator and a left parenthesis.

Arithmetic Operators

Binary Arithmetic Operator	Meaning
-	Addition
*	Subtraction
/	Multiplication
	Division
**	Exponentiation

Unary Arithmetic Operator	Meaning
+	The effect of multiplication by the numeric literal +1
-	The effect of multiplication by the numeric literal -1

Concatenation operator

The concatenation operator is the COBOL character '&', which shall be immediately preceded and followed by a separator space.

Relational operators

The relational operators specify the type of comparison to be made in a relation condition. A space shall precede and follow each reserved word of the relational operator. The relational operators meaning 'greater than or equal to' and 'less than or equal to' are extended relational operators. All other relational operators are simple relational operators. Use of the word NOT is prohibited within the string of words that represent an extended relational operator. When used, NOT is inserted into the string of words that comprise a simple relational operator, and that string, now including the word NOT, is one relational operator that defines the comparison to be executed for truth value.

Note - The relational operator IS NOT GREATER THAN is equivalent to IS LESS THAN OR EQUAL TO and IS NOT LESS THAN is equivalent to IS GREATER THAN OR EQUAL TO.

Relational Operator

Meaning	Relational Operator
Greater than or not greater than	IS [NOT] GREATER THAN IS [NOT] >
Less than or not less than	IS [NOT] LESS THAN IS [NOT] <
Equal to or not equal to	IS [NOT] EQUAL TO IS[NOT] =
Greater than or equal to	IS GREATER THAN OR EQUAL TO IS >=
Less than or equal to	IS LESS THAN OR EQUAL TO IS <=

Expressions

Arithmetic expressions

An arithmetic expression may be an identifier of a numeric elementary data item or a function, a numeric literal, the figurative constant ZERO (ZEROS, ZEROES), such identifiers, figurative constants, and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Any arithmetic expression may be preceded by a unary operator. The permissible combinations of identifiers, numeric literals, arithmetic operators, and parentheses are given in table 3, Combination of symbols in arithmetic expressions.

Formation and evaluation rules for arithmetic expressions depend on whether the mode of arithmetic in effect is native or standard.

Arithmetic

The following rules shall apply regardless of the mode of arithmetic that is in effect:

1. Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first, and, within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:

   1st - Unary plus and minus

   2nd - Exponentiation

   3rd - Multiplication and division

   4th - Addition and subtraction

2. Parentheses are used to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear, to modify the normal hierarchical sequence of execution in expressions where it is necessary to have some deviation from the normal precedence, or to emphasize the normal sequence for the sake of clarity. When the sequence of execution is not specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right.

3. The ways in which identifiers, literals, operators, and parentheses may be combined in an arithmetic expression are summarized in table 3, Combinations of symbols in arithmetic expressions, where:

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

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

Combinations of symbols in arithmetic expressions

	Second symbol	**Second symbol**	**Second symbol**	**Second symbol**	**Second symbol**
First symbol	**Identifier or Literal**	**+ - * / ****	**Unary + or -**	**(**	**)**
Identifier or literal	-	P	-	-	P
+ - * / **	P	-	P	P	-
Unary + or -	P	-	-	P	-
(	P	-	P	P	-
)	-	P	-	-	P

4. An arithmetic expression may begin only with the symbol '(', '+', 'B', an identifier, or a literal and may end only with a ')', an identifier, or a literal. There shall be a one-to-one correspondence between left and right parentheses of an arithmetic expression such that each left parenthesis is to the left of its corresponding right parenthesis. If the first operator in an arithmetic expression is a unary operator, it shall be immediately preceded by a left parenthesis if that arithmetic expression immediately follows an identifier or another arithmetic expression.

5. The following rules apply to evaluation of exponentiation in an arithmetic expression:

a. If the value of an expression to be raised to a power is zero, the exponent shall have a value greater than zero. Otherwise, the size error condition is raised.

b. If the evaluation yields both a positive and a negative real number, the value returned as the result is the positive number.

c. If the value of an expression to be raised to a power is less than zero, the evaluation of the exponent shall result in an integer. Otherwise, the size error condition is raised.

6. Arithmetic expressions allow the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items.

Native arithmetic

Temporary arithmetic items are stored in arbitrary precision arithmetic items. Truncation or rounding occurs at the last step whenever multi-step arithmetic occurs.

Concatenation expressions

A concatenation expression consists of two operands separated by the concatenation operator.

General format

{ literal-1 | concatenation-expression-1} & literal-2

Syntax rules

Both operands shall be of the same class, either alphanumeric or national, except that a figurative constant may be specified as one or both operands. Neither literal-1 nor literal-2 shall be a figurative constant that begins with the word 'ALL'.

Literal-2 cannot be a national literal.

General rules

1. The class of the concatenation expression resulting from the concatenation operation shall be:
   a. when one of the operands is a figurative constant, the class of the literal or concatenation expression that constitutes the other operand, or
   b. when both of the operands is a figurative constant, the class alphanumeric, or
   c. the same class as the operands.
2. The value of a concatenation expression shall be the concatenation of the value of the literals, figurative constants, and concatenation expressions of which it is composed.

3. A concatenation expression shall be equivalent to a literal of the same class and value, and may be used anywhere a literal of that class may be used.

Conditional expressions

Conditional expressions identify conditions that are tested to enable the object program to select between alternate paths of control depending upon the truth value of the condition. A conditional expression has a truth value represented by either true or false. Conditional expressions are specified in the EVALUATE, IF, PERFORM, and SEARCH statements. There are two categories of conditions associated with conditional expressions: simple conditions and complex conditions. Each may be enclosed within any number of paired parentheses, in which case its category is not changed.

Simple conditions

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

Relation conditions

A relation condition specifies a comparison of two operands. The relational operator that joins the two operands specifies the type of comparison. A relation condition shall have a truth value of 'true' if the specified relation exists between the two operands, and a truth value of 'false' if the relation condition does not exist.

A relation condition involving operands of category data-pointer is a data-pointer relation condition; a relation condition involving operands of category program-pointer is a program-pointer relation condition; otherwise, the relation condition is a general relation condition.

Comparisons are defined for the following:

1. Two operands of class numeric.
2. Two operands of class alphabetic.
3. Two operands of class alphanumeric.
4. Two operands of class national.
5. Two operands where one is a numeric integer and the other is class alphanumeric or national.
6. Two operands of different classes where each operand is from the set of classes alphanumeric, alphabetic, or national.
7. Comparisons involving indexes or index data items.
8. Comparisons involving two object reference identifiers.
9. Two operands of class pointer where each operand is of the same category.

For purposes of comparison, a group item shall be treated as an elementary alphanumeric data item. A class alphabetic operand shall be treated as though it were an operand of class alphanumeric.

The first operand is called the subject of the condition; the second operand is called the object of the condition. A relation condition shall contain at least one reference to a data item or function.

General format

Format 1 (general-relation):

{ identifier-1 | literal-1 | arithmetic-expression-1 | index-name-1 }

relop

{ identifier-2 | literal-2 | arithmetic-expression-2 | index-name-2 }

where relop is:

{IS [NOT] GREATER THAN | IS [NOT] > | IS [NOT] LESS THAN | IS [NOT] < | IS [NOT] EQUAL TO | IS [NOT] = | IS GREATER THAN OR EQUAL TO | IS >= | IS LESS THAN OR EQUAL TO | IS <=}

Format 3 (pointer or object):

identifier-3 { IS [NOT] EQUAL TO | IS [NOT] = } identifier-4

Format 4 (object instance of)

object-identifier-5 IS class-name-1

Syntax rules

FORMAT 1

Identifier-3 and identifier-4 shall reference data items of class pointer or object, and both shall be of the same category.

FORMAT 4

Evaluates as true if object-identifier-5 is an instance of class-name-1.

Comparison of numeric operands

For operands whose class is numeric, a comparison is made with respect to the algebraic value of the operands regardless of the manner in which their usage is described. The length of the literal or arithmetic expression operands, in terms of the number of digits represented, is not significant. Zero is considered a unique value regardless of the sign. When standard arithmetic is in effect, the number of digits of the standard intermediate data item used and whether there is rounding shall be as specified in the rounding rules for standard arithmetic. (See Rounding rules.)

Comparison of a numeric integer operand with an operand of class alphanumeric or national

The numeric integer operand shall be an integer literal or an integer numeric data item of the usage display or national. The other operand may be a literal or data item of class alphanumeric or national.

The integer is treated as though it were moved to an elementary data item of the same length as the integer, in terms of standard data format characters, and of the same class and usage as the comparand. Comparison then proceeds by the rules for comparison of two operands of the class of the comparand.

Comparison of mixed alphanumeric or national operands

An operand of class alphanumeric or national may be compared to another operand of class alphanumeric or national. When the classes of the operands differ, the alphanumeric operand is treated as though it were converted and moved in accordance with the rules of the MOVE statement to an elementary data item of class national with the same length in terms of character positions as the alphanumeric operand. Comparison then proceeds by the rules for comparison of two operands of class national.

Comparison of alphanumeric operands

An operand of class alphanumeric may be compared to another operand of class alphanumeric or to another operand treated as class alphanumeric for the purposes of comparison. Comparison is made with respect to the collating sequence of characters specified for the current alphanumeric program collating sequence. The length of an operand is the number of alphanumeric character positions in the operand. There are two cases to consider: operands of equal length and operands of unequal length.

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

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

2. Operands of unequal length. If the operands are of unequal length, comparison proceeds as though the shorter operand were extended on the right by sufficient alphanumeric spaces to make the operands of equal length. The preceding rules for operands of equal length then apply.

Comparison of national operands

An operand of class national may be compared with another operand of class national. Comparison is made with respect to the collating sequence of characters specified for the current national program collating sequence. The length of an operand is the number of national character positions in the operand. There are two cases to consider: operands of equal length and operands of unequal length.

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

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

2. Operands of unequal length. If the operands are of unequal length, comparison proceeds as though the shorter operand were extended on the right by sufficient national spaces to make the operands of equal length. The preceding rules for operands of equal length then apply.

Comparisons involving index-names and/or index data items

Relation tests may be made only between

1. two index-names. The result is the same as if the corresponding occurrence numbers were compared.
2. an index-name and a numeric data item or numeric literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal.
3. an index data item and an index-name or another index data item. The actual values are compared without conversion.

Comparisons between object reference identifiers

An object reference identifier shall only be compared with another object reference identifier, or one of the predefined object identifiers defined in, Predefined object identifiers.

Comparison of an object reference identifier shall be only with the relational operators for equality and inequality.

For object reference identifiers, the relation 'object-identifier-1 = object-identifier-2' has a true value if, and only if, the object identified by object-identifier-1 is the same object as the object identified by object-identifier-2.

Comparison of pointer operands

The operands are equal if they reference the same address.

Class condition

The class condition determines whether an operand is numeric, alphabetic, alphabetic-lower, alphabetic-upper, Boolean, or contains only the characters in the set of characters specified by the CLASS clause as defined in the SPECIAL-NAMES paragraph of the environment division.

General format

identifier-1 IS [NOT] { NUMERIC | ALPHABETIC | ALPHABETIC-LOWER | ALPHABETIC-UPPER | class-name-1}

Syntax rules

1. If the NUMERIC phrase is specified, identifier-1 shall reference a data item whose usage is display or national or whose category is numeric.
2. If the NUMERIC phrase is not specified, identifier-1 shall reference a data item whose usage is display or national. If identifier-1 is a function-identifier, it shall reference an alphanumeric or national function.

3. NUMERIC shall not be specified if the category of identifier-1 is alphabetic or if identifier-1 is a group item composed of elementary items whose data description indicates the presence of operational sign(s).
4. ALPHABETIC, ALPHABETIC-LOWER, or ALPHABETIC-UPPER shall not be specified if the category of identifier-1 is Boolean or numeric.

General rules

1. When the class condition does not include the word NOT and identifier-1 is a zero-length group item, the result of the class test is always false.
2. When the class condition includes the word NOT and identifier-1 is a zero-length group item, the result of the class test is always true.
3. If identifier-1 is not a zero-length group item, the truth value of the class condition without the word NOT is determined as follows:
   a. If NUMERIC is specified,
      1. If the category of identifier-1 is numeric,

         - If the usage of identifier-1 is implicitly or explicitly display or national, the condition is true if the presence or absence of an operational sign in the content of identifier-1 is in agreement with the data description of identifier-1 and if the content, except for the operational sign, consists entirely of the characters 0, 1, 2, 3, ..., 9. Valid operational signs are defined in 13.17.48, SIGN clause.

         - If the usage of identifier-1 is not display or national, the condition is true if the content of identifier-1 consists entirely of a valid representation for the usage and, if a PICTURE clause is specified, its numeric value is within the range of values implied by the PICTURE clause.

      2. If the category of identifier-1 is not numeric, the condition is true if the content of identifier-1 consists entirely of the characters 0, 1, 2, 3, ..., 9.

         - If ALPHABETIC is specified, the condition is true if the content of the data item referenced by identifier-1 consists entirely of the uppercase letters A, B, C, ..., Z, space, or the lowercase letters a, b, c, ..., z, space, or any combination of the uppercase and lowercase letters and spaces.

         - If ALPHABETIC-LOWER is specified, the condition is true if the content of the data item referenced by identifier-1 consists entirely of the lowercase letters a, b, c, ..., z, and space.

         - If ALPHABETIC-UPPER is specified, the condition is true if the content of the data item referenced by identifier-1 consists entirely of the uppercase letters A, B, C, ..., Z, and space.

         - If class-name-1 is specified, the condition is true if the content of the data item referenced by identifier-1 consists entirely of the characters listed in the definition of class-name-1 in the SPECIAL-NAMES paragraph.
4. If the word NOT is specified, the truth value is reversed.

Condition-name condition (conditional variable)

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with condition-name-1.

General Format

condition-name-1

Rules

1. If condition-name-1 is associated with a range or ranges of values, then the conditional variable is tested to determine whether or not its value falls in this range, including the end values.
2. The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.
3. The result of the test is true if one of the values corresponding to condition-name-1 equals the value of its associated conditional variable.

Switch-status condition

A switch-status condition determines the on or off status of an implementor-defined external switch. The switch-name and the on or off value associated with the condition shall be named in the SPECIAL-NAMES paragraph of the environment division.

General format

Rules

The result of the test is true if the switch is set to the specified position corresponding to condition-name-1.

Sign condition

The sign condition determines whether or not the algebraic value of an arithmetic expression is less than, greater than, or equal to zero.

General format

arithmetic-expression-1 IS [NOT] { POSITIVE | NEGATIVE | ZERO }

Rules

When used, NOT and the next key word specify one sign condition that defines the algebraic test to be executed for truth value. An operand is positive, if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero.

NOTE - NOT ZERO is a truth test for a nonzero (positive or negative) value.

Complex conditions

A complex condition is formed by combining simple conditions and/or complex conditions with logical connectors (logical operators 'AND' and 'OR') or by

negating these conditions with logical negation (the logical operator 'NOT'). The truth value of a complex condition, whether parenthesized or not, is the truth value that results from the interaction of the stated logical operators on its constituent conditions.

Logical operators meanings

Logical Operator	Meaning
AND	Logical conjunction; the truth value is true if both of the conjoined conditions are true; false if one or both of the conjoined conditions is false.
OR	Logical inclusive OR; the truth value is true if one or both of the included conditions is true; false if both included conditions are false.
NOT	Logical negation or reversal of truth value; the truth value is true if the condition is false; false if the condition is true.

The logical operators shall be preceded by a space and followed by a space.

Negated conditions

A condition is negated by use of the logical operator 'NOT' that reverses the truth value of the condition to which it is applied. Including a negated condition in parentheses does not change its truth value.

NOTE - The truth value of a negated condition is true if the truth value of the condition being negated is false; the truth value of a negated condition is false if the truth value of the condition being negated is true.

General format

NOT condition-1

Combined conditions

A combined condition results from connecting conditions with one of the logical operators 'AND' or 'OR'.

General format

condition-1 { {AND|OR} condition-2 } …

Precedence of logical operators and the use of parentheses

In the absence of the relevant parentheses in a complex condition, the precedence (i.e., binding power) of the logical operators determines the conditions to which the specified logical operators apply and implies the equivalent parentheses. The order of precedence is 'NOT', 'AND', 'OR'.

NOTE 1 Specifying 'condition-1 OR NOT condition-2 AND condition-3' implies and is equivalent to specifying 'condition-1 OR ((NOT condition-2) AND condition-3)'.

Where parentheses are used in a complex condition, they determine the binding of conditions to logical operators. Parentheses may, therefore, be used to depart from the normal precedence of logical operators as specified above. (See Order of evaluation of conditions.)

NOTE 2 The example complex condition above may be given a different meaning by specifying it as '(condition-1 OR (NOT condition-2)) AND condition-3'.

Table 5, Combinations of conditions, logical operators, and parentheses, indicates the ways in which conditions and logical operators may be combined and parenthesized. There shall be a one-to-one correspondence between left and right parentheses such that each left parenthesis is to the left of its corresponding right parenthesis.

Combinations of conditions, logical operators, and parentheses

	In a conditional expression:	**In a conditional expression:**	**In a left-to-right sequence of elements:**	**In a left-to-right sequence of elements:**
Given the Following element:	**May element be first?**	**May element be last?**	**Element, when not first, may be immediately preceded by only:**	**Element, when not last, may be immediately followed by only:**
simple-condition	Yes	Yes	OR, NOT, AND, (	OR, AND,)
OR or AND	No	No	Simple-condition,)	Simple-condition, NOT, (
NOT	Yes	No	OR, AND, (	Simple-condition, (
(	Yes	No	OR, NOT, AND, (	Simple-condition, NOT, (
)	No	Yes	Simple-condition,)	OR, AND,)

NOTE - The element pair 'OR NOT' is permissible while the pair 'NOT OR' is not permissible; the pair 'NOT (' is permissible while the pair 'NOT NOT' is not permissible

Abbreviated combined relation conditions

When simple or negated simple relation conditions are combined with logical connectives in a consecutive sequence such that a succeeding relation condition contains a subject or subject and relational operator that is common with the preceding relation condition, and no parentheses are used within such a consecutive sequence, any relation condition except the first may be abbreviated by:

1. The omission of the subject of the relation condition, or
2. The omission of the subject and relational operator of the relation condition.

Within a sequence of relation conditions, both forms of omission may be used.

General format

Relation-condition-1
{
{ AND | OR } { NOT | [NOT] simple-relational-operator | extended-relational-operator}
object-1
} ...

Syntax Rules

1. Relation-condition-1 shall not be a Boolean relation condition.
2. The result of implied insertion shall comply with the rules of table 5, Combinations of conditions, logical operators, and parentheses.
3. The word NOT shall not be followed immediately by the word NOT or the words IS NOT.

General rules

1. The effect of using abbreviations is as if the last preceding stated subject were inserted in place of the omitted subject, and the last stated relational operator were inserted in place of the omitted relational operator. The insertion of an omitted subject and/or relational operator terminates once a complete simple condition is encountered within a complex condition.
2. The interpretation applied to the use of the word NOT in an abbreviated combined relation condition is as follows:
   a. If an extended relational operator immediately follows the word NOT, then the NOT is interpreted as a logical operator; otherwise,
   b. If the relational operator following the word NOT is a simple relational operator, then the NOT participates as part of the simple relational operator; otherwise,
   c. The NOT is interpreted as a logical operator and, therefore, the implied insertion of subject or relational operator results in a negated relation condition.

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

Abbreviated combined relation condition

Abbreviated combined relation condition	Expanded equivalent
a > b AND NOT < c OR d	((a > b) AND (a NOT < c)) OR (a NOT < d)
a NOT EQUAL b OR c	(a NOT EQUAL b) OR (a NOT EQUAL c)
NOT a = b OR c	(NOT (a = b)) OR (a = c)
NOT (a GREATER b OR < c)	NOT ((a GREATER b) OR (a < c))
NOT (a NOT > b AND c AND NOT d)	NOT (((a NOT > b) AND (a NOT > c)) AND (NOT (a NOT > d)))

Order of evaluation of conditions

Parentheses, both explicit and implicit, denote a level of inclusiveness within a complex condition. Two or more conditions connected by only the logical operator 'AND' or only the logical operator 'OR' at the same level of inclusiveness establish a hierarchical level within a complex condition. Thus, an entire complex condition may be considered to be a nested structure of hierarchical levels with the entire complex condition itself being the most inclusive hierarchical level. Within this context, the evaluation of the conditions within an entire complex condition begins at the left of the entire complex condition and proceeds according to the following rule recursively applied where necessary:

1. The constituent connected conditions within a hierarchical level are evaluated in order from left to right, and evaluation of that hierarchical level terminates as soon as a truth value for it is determined regardless of whether all the constituent connected conditions within that hierarchical level have been evaluated.
2. Values are established for arithmetic expressions and 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. (See Arithmetic expressions.)

8. Input/output Files

File attributes

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

IOCS and MSCS

The processing of files is done by the Elastic COBOL runtime. This system is the input-output control system (IOCS). A subset of this system is used to process files that reside on mass storage. This is called the mass storage control system (MSCS). Since relative and indexed files are required to be mass storage resident, the processing of these files is done by the MSCS. Sequential files can reside on mass storage or other media, so the processing of these files is by the MSCS or Elastic COBOL virtual devices.

Organization

There are three file organizations: sequential, relative, and indexed.

Sequential

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

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

Relative

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

Conceptually, a file with relative organization is a serial string of areas, each capable of holding a logical record. Each of these areas is denominated by a relative record

number. Each logical record in a relative file is identified by the relative record number of its storage area. For example, the tenth record is the one addressed by relative record number 10 and is in the tenth record area, whether or not records have been written in any of the first through the ninth record areas.

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

Indexed

A file with indexed organization is a mass storage file from which any record may be accessed by giving the value of a specified key in that record. For each key data item defined for the records of a file, an index is maintained.

Each such index represents the set of values from the corresponding key data item in each record. Each index, therefore, is a mechanism that may provide access to any record in the file.

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

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

Both the prime record and any alternate record keys are made up from one or more portions of the record area associated with the file. For each key, the number of such components and their relative position within the record area is a fixed file attribute, and cannot be changed once the file has been created.

Access modes

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

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

Sequential access mode

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

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

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

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

Random access mode

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

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

Dynamic access mode

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

Reel and unit

The terms ‘reel’ and ‘unit’ are synonymous and completely interchangeable. The treatment of sequential mass storage files is logically equivalent to the treatment of a file on tape or analogous sequential media. Treatment of a file contained in a multiple tape file environment is logically equivalent to the treatment of a sequential single-unit file if the file is wholly contained on one unit.

Reel and unit clauses are used for IO status only.

Current volume pointer

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

File position indicator

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

I-O status

The I-O status is a two-character conceptual entity whose value is set to indicate the status of an input-output operation during the execution of a CLOSE, DELETE, OPEN, READ, REWRITE, START, UNLOCK or WRITE statement and prior to the execution of any imperative statement associated with that input-output statement or prior to the execution of any applicable USE EXCEPTION procedure. The value of the I-O status is made available to the program through the use of the FILE STATUS clause in the file control entry for the file.

The I-O status also determines whether an applicable USE EXCEPTION procedure will be executed. If any condition other than those listed below under the heading 'Successful Completion' results, such a procedure may be executed depending on rules stated elsewhere. If one of the conditions listed under the heading 'Successful Completion' results, no such procedure will be executed.

Certain classes of I-O status values indicate critical error conditions. These are: any that begin with the digit 3 or 4. Upon critical errors, Elastic COBOL chooses to continue execution of the run unit, and control is transferred to the end of the input-output statement that produced the critical error condition. Any NOT AT END or NOT INVALID KEY phrase specified for that statement is ignored.

I-O status expresses one of the following conditions upon completion of the input-output operation:

1. **Successful completion.** The input-output statement was successfully executed.
2. **At end.** A sequential READ statement was unsuccessfully executed as a result of an at end condition.
3. **Invalid key.** The input-output statement was unsuccessfully executed as a result of an invalid key condition.
4. **Permanent error.** The input-output statement was unsuccessfully executed as the result of an error that precluded further processing of the file. Any specified exception procedures are executed. The permanent error condition remains in effect for all subsequent input-output operations on the file unless an implementor-defined technique is invoked to correct the permanent error condition.
5. **Logic error.** The input-output statement was unsuccessfully executed as a result of an improper sequence of input-output operations that were performed on the file or as a result of violating a limit defined by the user.

6. **Record operation conflict.** The input-output statement was unsuccessfully executed as a result of the record being locked by another file connector.
7. **File sharing conflict.** The input-output statement was unsuccessfully executed as a result of the file being locked by another file connector.

The values placed in the I-O status for the previously named conditions resulting from the execution of an input-output operation. If more than one value applies, Elastic COBOL determines which of the applicable values to place in the I-O status.

Successful completion

1. **I-O status = 00.** The input-output statement is successfully executed and no further information is available concerning the input-output operation.
2. **I-O status = 04.** A READ statement is successfully executed but the length of the record being processed does not conform to the fixed file attributes for that file.
3. **I-O status = 05.** An OPEN statement is successfully executed but the referenced optional file is not present at the time the OPEN statement is executed. If the open mode is I-O or extend, the file has been created.
4. **I-O status = 07.** The input-output statement is successfully executed. However, for a CLOSE statement with the NO REWIND, REEL/UNIT, or FOR REMOVAL phrase or for an OPEN statement with the NO REWIND phrase, the referenced file is on a non-reel/unit medium.

At end condition with unsuccessful completion

1. **I-O status = 10.** A sequential READ statement is attempted and no next or prior logical record exists in the file because:
   a. NEXT was specified or implied and the end of the file has been reached, or
   b. PREVIOUS was specified and the beginning of the file has been reached, or
   c. a sequential READ statement is attempted for the first time on an optional input file that is not present.
2. **I-O status = 14.** A sequential READ statement is attempted for a relative file and the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file.

Invalid key condition with unsuccessful completion

1. **I-O status = 21.** 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 through a file connector and the execution of the next REWRITE statement for that file through the same file connector, or the ascending sequence requirements for successive record key values are violated. (See WRITE statement.)

2. **I-O status = 22.** An attempt is made either:
    a. to write a record that would create a duplicate key in a relative file.
    b. to write a record that would create a duplicate prime record key in an indexed file, or
    c. to write or rewrite a record that would create a duplicate alternate record key when the DUPLICATES phrase is not specified for that alternate record key in an indexed file.
3. **I-O status = 23.** This condition exists because:
    a. an attempt is made to randomly access a record that does not exist in the file; or
    b. a START or random READ statement is attempted on an optional input file that is not present.
4. **I-O status = 24.** An attempt is made to write beyond the externally defined boundaries of a relative or indexed file. The implementor specifies the manner in which these boundaries are defined. Or, a sequential WRITE statement is attempted for a relative file and the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file.

Permanent error condition with unsuccessful completion

1. **I-O status = 30.** A permanent error exists and no further information is available concerning the input-output operation.
2. **I-O status = 34.** A permanent error exists because of a boundary violation; an attempt is made to write beyond the externally defined boundaries of a sequential file. The implementor specifies the manner in which these boundaries are defined.
3. **I-O status = 35.** A permanent error exists because an OPEN statement with the INPUT, I-O, or EXTEND phrase is attempted on a non-optional file that is not present.
4. **I-O status = 37.** A permanent error exists because an OPEN statement is attempted on a file and that file will not support the open mode specified in the OPEN statement. The possible violations are:
    a. the EXTEND or OUTPUT phrase is specified but the file will not support write operations.
    b. the I-O phrase is specified but the file will not support the input and output operations that are permitted for the organization of that file when opened in the I-O mode.
    c. the INPUT phrase is specified but the file will not support read operations.
5. **I-O status = 38.** A permanent error exists because an OPEN statement is attempted on a file previously closed with lock.

6. **I-O status = 39.** The OPEN statement is unsuccessful because a conflict has been detected between the fixed file attributes and the attributes specified for that file in the program.

Logic error condition with unsuccessful completion

1. **I-O status = 41.** An OPEN statement is attempted for a file connector in the open mode.
2. **I-O status = 42.** A CLOSE or UNLOCK statement is attempted for a file connector not in the open mode.
3. **I-O status = 43.** For a mass storage file in the sequential access mode, the last input-output statement executed for the associated file through a file connector prior to the execution of a DELETE or REWRITE statement through the same file connector was not a successfully executed READ statement.
4. **I-O status = 44.** A boundary violation exists because:
   a. an attempt is made to write or rewrite a record that is larger than the largest or smaller than the smallest record allowed by the RECORD IS VARYING clause of the associated file-name, or
   b. an attempt is made to rewrite a record to a sequential file and the record is not the same size as the record being replaced.
   c. an attempt is made to write or rewrite a record that is larger than the largest or smaller than the smallest record allowed by the fixed-length or variable-length format of the RECORD clause when the implementor has specified that variable-length records are produced.
5. **I-O status = 46.** A sequential READ statement is attempted referencing a file connector open in the input or I-O mode and no valid next record has been established because:
   a. The preceding START statement referencing that file connector was unsuccessful, or
   b. The preceding READ statement referencing that file connector was unsuccessful.
6. **I-O status = 47.** The execution of a READ or START statement is attempted referencing a file connector that is not open in the input or I-O mode.
7. **I-O status = 48.** The execution of a WRITE statement is attempted referencing a file connector that is not open in the correct open mode as follows:
   a. If the access mode is sequential, the file connector is not open in the extend or output mode.
   b. If the access mode is dynamic or random, the file connector is not open in the I-O or output mode.
8. **I-O status = 49.** The execution of a DELETE or REWRITE statement is attempted referencing a file connector that is not open in the I-O mode.

Record operation conflict condition with unsuccessful completion

1. **I-O status = 54.** The input-output statement is unsuccessful because the statement requested a record lock, but this file connector holds the maximum number of locks allowed by Elastic COBOL (1000).
2. **I-O status = 99.** The input-output statement is unsuccessful because a required record could not be locked. Note that records other than the one specified may be required in order to obtain the requested record.

Extended File Status Codes

1. **I-O status = 90.** Not seekable. Random access is required for the operation, but not available for device. AS/400 error.
2. **I-O status = 91.** Cancelled. User cancelled the OPEN.
3. **I-O Status = 92.** Insufficient license. Elastic COBOL license in not sufficient for operation.
4. **I-O Status = 93.** Network error occurred in network layer.
5. **I-O Status = 95.** READ lock in indexed file.
6. **I-O Status = 96.** WRITE lock in indexed file.
7. **I-O Status = 97.** Not in Educational. Operation not supported with Educational Version.

File sharing conflict condition with unsuccessful completion

I-O status = 93. A file sharing conflict condition exists because an OPEN statement is attempted on a file and that file is already open by another file connector in a manner that conflicts with this request. The possible violations are:

a. An attempt is made to open a file that is currently open by another file connector in the sharing with no other mode.

b. An attempt is made to open a file in the sharing with no other mode and the file is currently open by another file connector.

c. An attempt is made to open a file for I-O or extend and the file is currently open by another file connector in the sharing with read only mode.

d. An attempt is made to open a file in the sharing with read only mode and the file is currently open by another file connector in the I-O or extend mode.

e. An attempt is made to open a file in the output mode and the file is currently open.

Invalid key condition

The invalid key condition may occur as a result of the execution of a DELETE, READ, REWRITE, START, or WRITE statement. When the invalid key condition occurs, execution of the input-output statement that recognized the condition is unsuccessful and the file is not affected.

If the invalid key condition exists after the execution of the input-output operation specified in an input-output statement, the following actions occur in the order shown:

1. The I-O status of the file connector associated with the statement is set to a value indicating the invalid key condition.
2. If the INVALID KEY phrase is specified in the input-output statement, any USE EXCEPTION file procedure associated with the file connector is not executed and control is transferred to the imperative-statement specified in the INVALID KEY phrase. Execution then continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the INVALID KEY phrase, control is transferred to the end of the input-output statement and the NOT INVALID KEY phrase, if specified, is ignored.
3. If the INVALID KEY phrase is not specified in the input-output statement and a USE AFTER EXCEPTION procedure is associated with the file connector associated with the input-output statement, the USE AFTER EXCEPTION procedure is executed and control is transferred according to the rules of the USE statement. The NOT INVALID KEY phrase is ignored, if it is specified.
4. If the INVALID KEY phrase is not specified in the input-output statement and there is no USE AFTER EXCEPTION procedure associated with the file connector associated with the input-output statement, control is transferred to the end of the input-output statement. The NOT INVALID KEY phrase is ignored, if it is specified.

If the invalid key condition does not exist after the execution of the input-output operation specified by an input-output statement, the INVALID KEY phrase is ignored, if specified. The I-O status of the file connector associated with the statement is updated and the following actions occur:

1. If the I-O status indicates an unsuccessful completion that is not an invalid key condition, control is transferred according to the rules of any USE EXCEPTION file procedure associated with the file connector.
2. If the I-O status indicates a successful completion, control is transferred to the end of the input-output statement or to the imperative-statement specified in the NOT INVALID KEY phrase if it is specified. In the latter case, execution continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the

imperative-statement specified in the NOT INVALID KEY phrase, control is transferred to the end of the input-output statement.

At end condition

The at end condition may occur as a result of the execution of a READ statement or a RETURN statement.

RETRY phrase

The RETRY phrase is specified in an input-output statement to indicate whether the MSCS should continue to attempt to obtain access in the event that a file or record is locked.

General format

{TIMEOUT RETRY arithmetic-expression-1 TIMES} |
{TIMEOUT RETRY {(NO LIMIT TIMES)|{ FOREVER}}|
{TIMEOUT AFTER arithmetic-expression-2 SECONDS}

General rules

1. Arithmetic-expression-1 specifies the number of times after the initial failure that the MSCS shall attempt to gain access to the locked resource and complete the requested input-output operation. There is a 500ms delay between client attempts. The value of arithmetic-expression-1 shall be a positive integer. If the NO LIMIT phrase is specified, these attempts shall continue until the input-output operation has been completed.
2. Arithmetic-expression-2 specifies the number of seconds in the timeout period. The I-O statement behaves as though the length of the timeout period were stored in a temporary data item whose picture is 9(n)V9(m), in the manner specified by this rule. The temporary data item is treated as double-precision floating-point so any valid COBOL number is permissible. There is no set maximum time limit. Arithmetic-expression-2 is used as the sending item and the temporary data item as the receiving item in a COMPUTE statement without the ROUNDED phrase. During the timeout period, the MSCS shall attempt to gain access to the locked resource and complete the requested input-output operation. There is a 500ms delay between client attempts.
3. If the I/O operation is unsuccessful on the first attempt because of a file sharing conflict condition or a record operation conflict condition, the following apply:
    a. If the RETRY phrase is not specified or the result of the evaluation of arithmetic-expression-1 or arithmetic-expression-2 is negative or zero, the statement is unsuccessful and a value is placed into the I-O status associated with the file connector to indicate a file sharing conflict condition or a record operation conflict condition; otherwise,
    b. The MSCS attempts to complete the input-output operation as specified in general rules.

If the MSCS permits the requested access on one of these attempts, the statement is successful and the results are as if the file sharing or record operation conflict had never occurred.

Otherwise, the statement is unsuccessful and a value is placed into the I-O status associated with the file connector to indicate a file sharing conflict condition or a record operation conflict condition. If there is an applicable declarative, it is executed. This may cause its own transfer of control; otherwise, execution continues with the statement following the input-output statement.

Sharing mode

The sharing mode indicates whether a file is to participate in file sharing and record locking, and specifies the degree of file sharing (or non-sharing) to be permitted for the file. The sharing mode specifies the types of operations that may be performed on the shared file through other file connectors throughout the duration of this OPEN.

The SHARING phrase on an OPEN statement overrides the SHARING clause in the file control entry for establishing the sharing mode. If there is no SHARING phrase on the OPEN statement, the sharing mode is completely determined by the SHARING clause in the file control entry. If no specification is made in either location, no explicit sharing occurs; the operating systems rules preempt. The rules are the same for a given sharing mode regardless of whether the sharing mode is specified on the open statement or specified in the file control paragraph.

Other facilities, such as a job control language or another programming language, may specify some degree of file sharing, however, if COBOL syntax is used to specify file sharing, file sharing is done as defined herein.

NOTE – Java does not normally possess record locking, so custom Elastic COBOL methods are used to implement the file sharing and record locking. The methods used for file sharing are documented below. The methods used for record locking are beyond the scope of this document.

A shared file shall reside on a device that allows concurrent access to the file. The device and permissions must support file renaming.

File sharing is done by means of file renaming. A file in the ‘base state’ (having the same name specified in its ASSIGN) is an inactive file, or one for which no file sharing semantics had been specified leaving its control to the operating system.

When an attempt to open a file occurs, it attempts to rename the file with a specific prefix depending on the sharing mode:

“no_” for NO OTHERS,

“aoi_” for ALL OTHERS opened initially for INPUT,

“aoo_” for ALL OTHERS opened initially for other than INPUT,

“roi_” for READ ONLY opened initially for INPUT,

“roo_” for READ ONLY opened initially for other than INPUT.

If the file renaming cannot occur, then it attempts only to open those file prefixes which are valid for its sharing condition. The file renaming is automatically undone

upon normal closure of the file. If a program abnormally terminates, files may be unintentionally left in the shared state; they may be renamed to remove the prefix to move them back to the inactive unshared state.

Note that the EXCLUSIVE and UN-EXCLUSIVE verbs use the record server to achieve file sharing instead of the above renaming scheme due to the EXCLUSIVE and UN-EXCLUSIVE verbs working after the file is opened instead of at the time of opening.

Before access to a shared file is allowed through an OPEN statement, the sharing mode and the open mode shall be allowed by all other file connectors that are currently associated with the file. Additionally, the sharing mode for the current OPEN statement shall permit all of the sharing modes and open modes that exist for all other file connectors that are currently associated with the file. (See I-O status; OPEN statement; and table Opening available shared files that are currently open by another file connector.)

The sharing mode controls access to a file as follows:

1. The sharing with no other mode specifies exclusive access to a file. Associating this file connector with the file will be unsuccessful if the file is currently open through other file connectors. If the OPEN statement is successful, subsequent requests to open the file through other file connectors before this file connector is closed will be unsuccessful. Record locks are ignored.
2. The sharing with read only mode restricts concurrent access to a file through file connectors other than this one, to input mode. Associating this file connector with the file will be unsuccessful if the file is currently open in a mode other than input. If the OPEN statement is successful, subsequent requests to open the file through other file connectors in a mode other than input before this file connector is closed will be unsuccessful. Record locks are in effect.
3. The sharing with all other mode allows concurrent access to a file through other file connectors specifying input, I-O, or extend mode, subject to any further restrictions that apply. Record locks are in effect.

Multiple paths of access may exist concurrently for a shared file whether they exist in the same program, contained programs, separately compiled programs within the same run unit, or separately compiled programs in different run units. Do not open the same file connector from the same run unit.

The setting of a file lock is part of the atomic operation of an I-O statement.

The file lock and all record locks established for a file connector are removed by an explicit or implicit CLOSE statement executed for that file connector.

Sort/Merge Files

Sort file

A sort file is a collection of records to be sorted by a SORT statement. The rules for blocking and for allocation of internal storage are peculiar to the SORT statement. The RELEASE and RETURN statements imply nothing with respect to buffer areas,

blocks, or reels. A sort file, then, may be considered as an internal file that is created (RELEASE statement) from the input file, processed (SORT statement), and then made available (RETURN statement) to the output file.

A sort file is named by a file control entry and is described by a sort-merge file description entry. A sort file is referred to by the RELEASE, RETURN, and SORT statements.

Sort files are automatically removed upon completion of the sort.

Merge file

A merge file is a collection of records to be merged by a MERGE statement. The merge file has no label procedures that the programmer may control and the rules for blocking and for allocation of internal storage are peculiar to the MERGE statement. The RETURN statement implies nothing with respect to buffer areas, blocks, or reels. A merge file, then, may be considered as an internal file that is created from input files by combining them (MERGE statement) as the file is made available (RETURN statement) to the output file.

A merge file is named by a file control entry and is described by a sort-merge file description entry. A merge file is referred to by the RETURN and MERGE statements.

Merge files are automatically removed upon completion of the merge.

Screens

Terminal screen

A terminal provides I-O via a screen, which is a rectangular array of displayable character locations, and a keyboard.

A screen contains one or more fields during each input or output operation. A field may range in size from one character to the maximum number of characters permitted on the screen. Each field represents an elementary screen item. One or more fields may be logically grouped together into a group screen item; such fields need not be contiguous. A group screen item may contain other group screen items. The fields within a group screen item are ordered for the purposes of determining the next field and the previous field operations during terminal input. The order of fields is determined by the order of declaration of screen items in a screen description entry.

A screen has visible attributes associated with each display location.

The Elastic COBOL CONSOLE device must be used for full screen operations. On graphical systems, the CONSOLE is graphical in nature; on text systems, the text terminal is used. The graphical CONSOLE defaults to 80 columns by 25 lines. The SYSOUT and SYSIN devices may not be used for full screen control.

Elastic COBOL allows both text and graphics on the CONSOLE device. Because of this, any graphical elements at the same location as text elements obscure text elements.

A default CONSOLE will be created upon first use. The attributes of the CONSOLE may be explicitly controlled by program code if the first DISPLAY is a DISPLAY STANDARD|GRAPHICAL WINDOW.

The default ACCEPT/DISPLAY device is CONSOLE unless, for example “DISPLAY xxx UPON SYSOUT” is specified on the statement. To change this ACCEPT/DISPLAY default to SYSIN/SYSOUT (standard input, standard output) use the Elastic COBOL compiler option **-run:system**.

CRT status

The CRT status is a three-character conceptual entity whose value is set to indicate the status of a terminal input-output operation during the execution of an ACCEPT screen statement and prior to the execution of any imperative statement associated with any ON EXCEPTION or NOT ON EXCEPTION clauses for that ACCEPT statement. The value of the CRT status is made available to the COBOL program through the use of the CRT status clause in the SPECIAL-NAMES paragraph. The third character is currently unused.

The CRT status follows the X/Open usage.

CRT status expresses one of the following conditions upon completion of the input operation:

1. Successful completion with normal termination. The input statement was successfully executed.
2. Successful completion with termination by a function key keystroke. The input statement was successfully executed.
3. Unsuccessful completion. The input statement was not successfully executed. Further terminal I-O statements are not precluded.

The following is a list of the values placed in the CRT status for the conditions resulting from the execution of an input operation.

1. The first character of the CRT Status data item is CRTKEY1, and the second character of the CRT Status data item is CRTKEY2.
2. If CRTKEY1 is ‘0’, then the input was terminated. If CRTKEY2 is ‘0’ then a terminator was pressed, if CRTKEY2 is ‘1’ then the user auto skipped out of the last field.
3. If CRTKEY1 is ‘1’, then a user defined function key was pressed. CRTKEY2 is a single byte character holding the binary number of the function key.
4. If CRTKEY1 is ‘2’, then a system defined function key was pressed. CRTKEY2 is a single byte character holding the binary number of the function key.
5. If CRTKEY1 is ‘9’, then no items fell within the screen.

Cursor

Character addressable terminals use the concept of a cursor to indicate the position on the screen at which keyboard operations will be displayed. This is generally indicated by the position of a visible cursor symbol.

During execution of a DISPLAY screen statement, the position and visibility of the cursor is undefined.

During execution of an ACCEPT screen statement, the position and visibility of the cursor is undefined except during the period that the keyboard is synchronously enabled for operator input. The cursor shall be visible during this period and shall indicate the position on the screen at which keyboard input will be displayed.

During execution of an ACCEPT screen statement, the cursor is initially positioned at the first elementary screen item in the screen description entry whose specification includes a TO or USING phrase, unless the CURSOR clause is specified in the SPECIAL-NAMES paragraph, in which case the cursor is positioned as specified in that clause.

Once the keyboard is enabled for operator input, the operator may use cursor positioning keys to move between elementary screen items whose specification includes a TO or USING clause. Depending on the screen description entry for the item, the operator may move the cursor between characters within the displayed item.

Left and Right cursor keys maneuver within the fields. Tab and Down cursor advance a field. Shift-Tab and Up go back a field. Home moves to the beginning of a field. End moves to the end of a field. Control-Home goes to the top field. Control-End goes to the bottom field.

On the GUI Console, a popup menu is available by right-clicking. The popup menu offers the ability to change the font size, columns and lines, background and foreground colors, and in Java 2 the ability to print the GUI Console.

On international systems in Java 2, input methods may be activated in the GUI Console to allow input of complex characters not readily handled by the standard keyboard. These may be assigned to national data items only; assigning national data to standard alphanumeric results in the truncation of each national character.

Cursor locator

The cursor locator is a four- or six-character conceptual entity whose value is set by the program to indicate the position of the visible cursor on the display screen when the keyboard becomes synchronously enabled during execution of an ACCEPT screen statement. The position is relative to the top left corner of the screen.

Upon successful termination of execution of an ACCEPT screen statement, the cursor locator is set to indicate the position of the visible cursor at the time the operator presses the terminator key or a function key. If the execution of the ACCEPT statement was unsuccessful, the value of the cursor locator is undefined.

The locator is split in half; for a four-character in two and two, for a six-character in three and three. The cursor locator is made available to the program through the

use of the CURSOR clause in the SPECIAL-NAMES paragraph. The first half represents a number giving the line number, the topmost line being 01 or 001. The second half represents a number giving the column number, the first column number being 01 or 001. If the position of the visible cursor is at a line or column number that is greater than 999, the value of the cursor locator is undefined.

Current screen item

During the execution of an ACCEPT statement, one or more elementary input screen items may be displayed on the terminal display. The operator is able to move the cursor between the screen items using context-dependent cursor positioning keys. The cursor may also move automatically from one screen item to another when the screen item becomes full or the last character in the screen item is keyed. The screen item in which the cursor is located is the current screen item. Any data keyed by the operator will be attributed to the current screen item and may cause the display of the current screen item to change. While a screen item is the current screen item, the display on the screen will not necessarily conform to the PICTURE clause of that item, but it will conform once the screen item is no longer current.

Color number

Color is one of the attributes that may be specified for screen items. For a monochrome terminal, the color attributes are mapped onto other attributes by Java and the operating system.

The colors available to a program are as in the following table. A color is selected by specifying the corresponding integer that represents the color number. The colors given in the table are a rough guide only, the exact shade of color will depend on the terminal capabilities. For example the value 6 might be shown as brown, but when HIGHLIGHT is also selected it might appear as yellow or the value 0 might be shown as black but when HIGHLIGHT is also selected it might appear as gray.

ScreenColor Numbers

Black	0
Blue	1
Green	2
Cyan	3
Red	4
Magenta	5
Brown or Yellow	6
White	7

In Elastic COBOL, the color names may be used after the keywords FOREGROUND-COLOR and BACKGROUND-COLOR instead of the color number. Additionally, bright- or light- may be appended to the front of the name to use the brighter color, or dim- or dark- may be appended to the front of the name to use the darker color. The color names are not reserved words.

Keystroke Variable

The KEYSTROKE variable defines keyboard control. The KEYSTROKE configuration variable may be set to text describing keyboard control. Each KEYSTROKE setting is additive to settings prior. The KEYSTROKE may be set outside of the program, as a program variable or in a configuration file, or it may be set in the program using SET CONFIGURATION "KEYSTROKE" TO value.

The KEYSTROKE has several commands that may be done when setting KEYSTROKE.

AT-END=value

DATA=value

EDIT=value

EXCEPTION=value

HOT-KEY=program-name

INVALID=value

TERMINATE=value

The format of the KEYSTROKE string is 'KEYSTROKE command… key-code'. The key-code is a code from the following list. Where applicable, the corresponding Java VK key code name is given.

Keystroke Variables

Key-Code	Meaning	Java VK Code
ZB	Backspace	VK_BACK_SPACE;
^M	Enter	VK_ENTER
^A-^Z	Control A-Z	Codes 1..26
k1	F1	VK_F1
k2	F2	VK_F2
k3	F3	VK_F3
k4	F4	VK_F4
k5	F5	VK_F5
k6	F6	VK_F6
k7	F7	VK_F7
k8	F8	VK_F8
k9	F9	VK_F9
k0	F10	VK_F10
k10	F10	VK_F10
k11	F11	VK_F11
k12	F12	VK_F12
kd	Cursor Down	VK_DOWN
kh	Home	VK_HOME
kl	Cursor Left	VK_LEFT
ku	Cursor Up	VK_UP
kA	Insert Line	VK_I + ALT_MASK
kB	Shift Tab	VK_TAB + SHIFT_MASK
kE	Clear to End	VK_END + CTRL_MASK
kL	Delete Line	VK_K + ALT_MASK
kN	Page Down	VK_PAGE_DOWN
kP		VK_PAGE_UP

Key-Code	Meaning	Java VK Code
K1		VK_F1 + SHIFT_MASK
K2		VK_F2 + SHIFT_MASK
K3		VK_F3 + SHIFT_MASK
K4		VK_F4 + SHIFT_MASK
K5		VK_F5 + SHIFT_MASK
K6		VK_F6 + SHIFT_MASK
K7		VK_F7 + SHIFT_MASK
K8		VK_F8 + SHIFT_MASK
K9		VK_F9 + SHIFT_MASK
K0		VK_F10 + SHIFT_MASK
K10		VK_F10 + SHIFT_MASK
K11		VK_F11 + SHIFT_MASK
K12		VK_F12 + SHIFT_MASK
Kc	Cancel	VK_X + ALT_MASK
Kd	Next Paragraph	VK_DOWN + CTRL_MASK
Kl	Word Left	VK_LEFT + CTRL_MASK
Kr	Word Right	VK_RIGHT + CTRL_MASK
Ku	Prev. Paragraph	VK_UP + CTRL_MASK
Kx	Exit	VK_E + ALT_MASK
KA	Attention	VK_A + ALT_MASK
KB	Bottom	VK_PAGE_DOWN + CTRL_MASK
KC	Clear	VK_HOME + CTRL_MASK
KD	Command(Do)	VK_D + ALT_MASK
KE	End	VK_END
KF	Find	VK_F + ALT_MASK
KI	Insert Char.	VK_INSERT
KL	Page Left	VK_L + ALT_MASK
KM	Mark	VK_M + ALT_MASK
KP	Print	VK_P + ALT_MASK
KR	Page Right	VK_R + ALT_MASK
KS	Send	VK_S + ALT_MASK
KT	Top	VK_UP + CTRL_MASK
KV	Save	VK_V + ALT_MASK
KX	Delete	VK_DELETE
K?	Help	VK_H + ALT_MASK
U1	User/Alt F1	VK_F1 + ALT_MASK
U2	User/Alt F2	VK_F2 + ALT_MASK
U3	User/Alt F3	VK_F3 + ALT_MASK
U4	User/Alt F4	VK_F4 + ALT_MASK
U5	User/Alt F5	VK_F5 + ALT_MASK
U6	User/Alt F6	VK_F6 + ALT_MASK
U7	User/Alt F7	VK_F7 + ALT_MASK
U8	User/Alt F8	VK_F8 + ALT_MASK
U9	User/Alt F9	VK_F9 + ALT_MASK
U0	User/Alt F10	VK_F10 + ALT_MASK
U10	User/Alt F10	VK_F10 + ALT_MASK
U11	User/Alt F11	VK_F11 + ALT_MASK
U12	User/Alt F12	VK_F12 + ALT_MASK
A1	Alt 1	VK_1 + ALT_MASK
A2	Alt 2	VK_2 + ALT_MASK
A3	Alt 3	VK_3 + ALT_MASK
A4	Alt 4	VK_4 + ALT_MASK
A5	Alt 5	VK_5 + ALT_MASK
A6	Alt 6	VK_6 + ALT_MASK
A7	Alt 7	VK_7 + ALT_MASK
A8	Alt 8	VK_8 + ALT_MASK
A9	Alt 9	VK_9 + ALT_MASK
A0	Alt 0	VK_0 + ALT_MASK

Key-Code	Meaning	Java VK Code
A-	Alt -	VK_SUBTRACT + ALT_MASK
A=	Alt =	VK_EQUALS + ALT_MASK
C1	Control F1	VK_F1 + CTRL_MASK
C2	Control F2	VK_F2 + CTRL_MASK
C3	Control F3	VK_F3 + CTRL_MASK
C4	Control F4	VK_F4 + CTRL_MASK
C5	Control F5	VK_F5 + CTRL_MASK
C6	Control F6	VK_F6 + CTRL_MASK
C7	Control F7	VK_F7 + CTRL_MASK
C8	Control F8	VK_F8 + CTRL_MASK
C9	Control F9	VK_F9 + CTRL_MASK
C0	Control F10	VK_F10 + CTRL_MASK
C10	Control F10	VK_F10 + CTRL_MASK
C11	Control F11	VK_F11 + CTRL_MASK
C12	Control F12	VK_F12 + CTRL_MASK
Num.	ASCII keycode	ASCII keycode

AT-END

If value starts with 'Y', 'y', 'T', 't', or '1' then the keycode becomes a termination keycode which also causes the AT END condition.

DATA

Assign special characters to the keycode.

EDIT

Assign a command value to the keycode from the following list:

Auto-Insert	First	Page-Down
Backspace	Home	Page-Up
Default-Entry	Insert-Off	Previous
Default-Next	Insert-On	Previous-All
Delete	Insert-Space	Previous-Line
Down	Last	Right
End	Menu	Switch-Window
Erase-All	Next	System-Menu
Erase-EOS	Next-All	Toggle-Edit-Mode
Erase Field	Next-Line	Toggle-Insert
Erase-Next	Numeric-Default	Up
Erase-to-End	Numeric-Next	

EXCEPTION

EXCEPTION creates an exception key. EXCEPTION makes the keycode terminate with the given integer exception value. TERMINATE and EXCEPTION is very similar in usage in Elastic COBOL .

HOT-KEY

HOT-KEY assigns an implicit action to the keycode, telling it to run the given program-name. The HOT-KEY command is currently ignored.

INVALID

If value starts with 'Y', 'y', 'T', 't', or '1' then the keycode is ignored when typed. If value starts with 'N', 'n', 'F', 'f', or '0' then the ignore is disabled.

TERMINATE

TERMINATE creates a termination key. TERMINATE makes the keycode terminate with the given integer termination value. TERMINATE and EXCEPTION is very similar in usage in Elastic COBOL.

SCREEN CONTROL

SCREEN CONTROL specifies a group identifier to be the screen control allowing an embedded procedure to control its containing ACCEPT statement.

Input fields in the screen section item being accepted are given sequentially ordered field numbers starting at one.

During an ACCEPT, if there is an embedded procedure, then ACCEPT-REASON is set to the reason for entry; this is 1 for notify event, 0 otherwise. ACCEPT-HANDLE is set to the handle of control for which the embedded procedure is executing; this is 0 for a non-graphical or textual control. ACCEPT-FIELD-NUMBER is set to the input field number of the control. ACCEPT-ID is set to the control's ID number if graphical, 0 if textual.

The SCREEN CONTROL group item is defined below. Any user SCREEN CONTROL item must follow this record format.

```
01 SCREEN-CONTROL-ITEM.
   05 ACCEPT-REASON-FOR-ENTRY PIC 9.
   05 ACCEPT-FIELD-NUMBER PIC 999.
   05 ACCEPT-HANDLE USAGE HANDLE.
   05 ACCEPT-ID PIC X(2) COMP-X.
```

When the ACCEPT terminates, ACCEPT-HANDLE is set to 0 and ACCEPT-FIELD-NUMBER contains the field number last containing the cursor.

During an embedded procedure, the ACCEPT-REASON may be set to alter the behavior of the remainder of the ACCEPT. The following settings are meaningful after the termination of the embedded procedure:

0	ACCEPT continues normally
1	ACCEPT continues from field ACCEPT-FIELD-NUMBER If ACCEPT-HANDLE is 0, then ACCEPT remains in current field. If ACCEPT-FIELD-NUMBER is set to a non-existent field, ACCEPT will terminate. If ACCEPT-FIELD-NUMBER is set to a read-only field, control passes to numerically next field number of terminates if there is none.
2	ACCEPT terminates normally, ACCEPT-FIELD-NUMBER is termination value of ACCEPT
3	ACCEPT terminates with exception, ACCEPT-FIELD-NUMBER is exception value of ACCEPT.
4	ACCEPT continues with control transferred to graphical control identified by ACCEPT-ID.

EVENT STATUS

EVENT STATUS specifies an identifier to receive the event status when a graphical screen section event occurs.

The EVENT STATUS group item is defined as follows:

```
01 EVENT-STATUS.
   05 EVENT-TYPE PIC X(4) COMP-X.
   05 EVENT-WINDOW-HANDLE USAGE HANDLE OF WINDOW.
   05 EVENT-HANDLE USAGE HANDLE.
   05 EVENT-ID PIC X(2) COMP-X.
   05 EVENT-DATA-1 PIC SX(2) COMP-X.
   05 EVENT-DATA-2 PIC SX(4) COMP-X.
   05 EVENT-ACTION PIC X COMP-X.
```

The EVENT-TYPE is identifies the type of the EVENT.

The EVENT-WINDOW-HANDLE holds the window of the control causing the event.

The EVENT-HANDLE holds the handle of the control that caused the event.

The EVENT-ID holds the ID of the control that caused the event.

The EVENT-DATA-1 holds event-specific information; see the specific event for more information.

The EVENT-DATA-2 holds event-specific information; see the specific event for more information.

The EVENT-ACTION holds the action that the event handler should perform upon termination of the event procedure. This value is from the following list:

0	EVENT-ACTION-NORMAL	Event processed normally, terminating only for terminating events.
1	EVENT-ACTION-TERMINATE	Event processed normally, and then always terminated.
2	EVENT-ACTION-CONTINUE	Event processed normally, and then always not terminated.
3	EVENT-ACTION-IGNORE	Event not processed further, not terminated.
4	EVENT-ACTION-FAIL	Specific setting for some events; event will indicate if this is used.
5	EVENT-ACTION-FAIL-	Fail and terminate.

	TERMINATE	

CURSOR

CURSOR specifies an identifier to control the cursor, where the identifier is 4 or 6 characters in length representing rrcc or rrrccc for row and column.

The CURSOR clause, if used, specifies the starting point for an ACCEPT.

The CURSOR clause, if used, will hold the final cursor position at the end of the ACCEPT.

CRT STATUS

CRT STATUS specifies an identifier for general CRT status.

The CRT STATUS identifier may be in one of two forms. The first form, compliant with X/Open and most COBOL compilers, is a three-character group. The second form, compliant with AcuCOBOL, is a numeric identifier which holds the termination or exception keycode upon completion of an ACCEPT.

The first form group has three characters, known as KEY-1, KEY-2 and KEY-3. KEY-3 holds the termination or exception key upon completion of an ACCEPT, the same keycode value which would be present if CRT STATUS referenced a numeric identifier. Below, keycode is X'00' to X'FF', a single digit binary (suitable as PIC X COMP-X).

CRT Status Table

KEY-1	KEY-2	KEY-3	Meaning
'0'	'0'	keycode	Termination key
'0'	'1'	N/A	Auto-skip out of last field
'1'	keycode	N/A	User-defined function/exception key
'2'	X'00'	N/A	Implementation-defined function/exception key
'9'	X'00'	N/A	No acceptable item on screen.

File Assignment and Protocols

An ASSIGN TO in Elastic COBOL assigns to the local file system normally. The ASSIGN TO filename should be a non-numeric literal such as "myfilename". Any local filenames are treated as Posix filenames; any file which Posix may address may be addressed by Elastic COBOL. Note, however, this does not imply that the data file storage format is compatible with Elastic COBOL. Because ANSI did not define the binary file storage format of files (especially indexed files), different vendors have different and incompatible file storage. Elastic COBOL does address certain other vendors' implementations, discussed below.

For some cross-platform file capability, Elastic COBOL defines a 'literal IS FILE CHARACTER' clause in the SPECIAL-NAMES paragraph. This allows the implied directory separator, such as '/' or '\' to be specified, allowing it to be automatically converted to the real file separator on the execution platform.

Elastic COBOL employs a file protocol method for assignment of files. An ASSIGN TO with a filename uses the default 'file' protocol which handles files on mass-storage devices (such as hard drives, CD's, floppies, etc.). All protocols are specified in the ASSIGN. The ASSIGN may be dynamically assigned to a variable, in which case the protocols may be STRINGed together dynamically as well.

In addition, Elastic COBOL has certain virtual devices available. This allows certain virtual devices, such as a TCP/IP socket or system clipboard, to be treated as a file by Elastic COBOL. This also allows normal files to be remotely accessible, using an intermediary (such as our remote file handler or NFS) to handle the file access. Many of these devices are Sequential Stream Devices, meaning that if INPUT-OUTPUT is allowed, REWRITE is not allowed (returning a not seekable I-O status), and data written is not necessary data which may be read; the semantics of the READ and WRITE are defined naturally for the device. Sequential Stream Devices may only be opened as Sequential Organization.

Also, Elastic COBOL supports multiple file formats for the sequential, relative, and indexed file types. As the binary storage format of these files is not defined by ANSI, different vendors have different flavors of storage. A file protocol may specify that a file be created and used in a non-standard or vendor-specific format.

Some protocols are additive, allowing other protocols to be added. In such a case, all file protocols are applied left to right. A file protocol 'remote:mf:filename' would mean access a remote file and then let the remote access be done in Micro Focus compatible format; this is because the remote: protocol knows about all the protocols of Elastic COBOL, and it is told to open 'mf:filename' which it understands. On the other hand, the NFS protocol knows nothing of Elastic COBOL protocols, so in 'nfs:filename' the filename must be comprehensible to the NFS filesystem; it could not be 'nfs:mf:filename'.

Acu:filename	AcuCOBOL format file. Sequential, Relative and Indexed. If using this protocol, it's recommended to compile using -Dca for AcuCOBOL data compatibility.
Mfu:filename	MicroFocus Unix format file. Sequential, Relative and Indexed. If using this protocol, it's recommended to compile using -Dcm for Micro Focus data compatibility.
Mfw:filename	Micro Focus Windows format file. Sequential, Relative and Indexed. If using this protocol, it's recommended to compile using -Dcm for Micro Focus data compatibility.
Mf:filename	Micro Focus format file (appropriate to current platform). Sequential, Relative and Indexed. If using this protocol, it's recommended to compile using -Dcm for Micro Focus data compatibility.
Isam:filename	ISAM format file. Indexed Only. This is a native indexed file format which requires a third-party native ISAM compliant driver such as D-ISAM.
@[host:]filename	AcuConnect Indexed Only. Requires AcuConnect be running on host. If host not specified, defaults to localhost. If running as an applet, host must be the webserver due to security.
Acon:[host:]filename	AcuConnect Indexed Only. Requires AcuConnect be running on host. If host not specified, defaults to localhost. If running as an applet, host must be the webserver due to security.
As400:[host:]filename	AS/400 Indexed Only. Requires IBM's AS/400 Java Toolbox in classpath (distributed with Elastic COBOL). If host not specified, defaults to localhost. If running as an applet, host must be the webserver due to security.

Clipboard:filename	Sequential Stream Only. The system clipboard, this is suitable for reading and writing textual application data sharable with other system programs. Filename is not used.
Copy:filename	Synonym for clipboard
Paste:filename	Synonym for clipboard
Console:filename	Graphical Console device referencable as a file. If filename is not used, this is the master Console. If filename is used, the named console is referenced; if there is no console by the given name, a new console is created. This allows multiple independent textual displays.
File:filename	Synonym for filename, a local mass-storage control file.
Http:url	Sequential Stream Only. The http: URL is used for opening an HTTP connection, allowing webpages and similarly served data to be handled directly. HTTP is generally capable of INPUT or OUTPUT, but not INPUT-OUTPUT. If running as an applet, host must be the webserver.
Ims:host:port	Sequential Stream Only. Connect to IMS at given TCP/IP address or host name and given port number. Allows READ and WRITE IMS data source across TCP/IP. If running as an applet, host must be the webserver.
Ipcio:filename	Sequential Stream Only. Filename is opened as a native program, and the data pipe to and from the program is used for READ and WRITE. This device uses the block size clause if present. Filename is in the form used at the command line.
Mailto:mail-recipient/sender=mail-sender/host=mail-host:mail-port	Sequential Stream Only. OUTPUT only. A mail message is sent to mail-recipient as if sent by mail-sender using the mail server at mail-host on port mail-port. A mail-server must be running at the given location. If running as an applet, mail-host must be the webserver for security reasons.
Nfs:nfs-filename	Network File System. The nfs-filename is passed through to WebNFS for Java; the third-party WebNFS plug-in from Sun Microsystems must be available in the CLASSPATH for this functionality. It must be a standard NFS filename usable by the remote NFS server; see the NFS server's documentation for what constitutes a valid NFS filename for the system. NFS may fully access Sequential and Relative files. It may read Indexed files; do not write Indexed files with it as the protocol does not recognize the indexed file format and therefore splits and maintenance must be done across the network instead of on the remote machine. Be aware that writing data through NFS is more likely to corrupt the data during a write; no special recovery is available for NFS during writes across the network. See remote: for more efficient remote access of indexed files. An NFS server must be running on the NFS host; this is supplied by the operating system vendor or third-party. If running as an applet, the NFS host must be the webserver for security reasons.
Printer: [FONT=fontname/] [SIZE=size/] [COLS=cols/] [ROWS=rows/] [ALIGNX=alignx/] [ALIGNY=aligny/] [MARGINX=marginx/] [MARGINY=marginy/] [BOLD][ITALIC]	Sequential Stream Only. As Java does not support text printing, this is a virtual printer device which renders text to an internal graphics display and then prints the graphics display. This implies that this printer will work for all graphical printers with a suitable operating system device driver, but that specific printer control sequences will be rendered as text, not as control sequences for the printer.
Remote:host:filename[:port]	Remote File System. The filename is passed through to the remote file system for further handling. As such, any format protocols (such as 'mf:' or 'acu:') should be within the filename. This protocol efficiently handles Sequential,

	Relative and Indexed files. The Remote File Server is provided with Elastic COBOL, and will run on any system for which Elastic COBOL is licensed; it must be running for the remote protocol to succeed. Automatic 'safe' record locking is automatically provided by the remote file system.
Server:port	Sequential Stream Only. A TCP/IP ServerSocket is created on the given port. An OPEN on the server socket waits for a connection. When a connection is received, the file reference actually refers to a standard Socket, not a Server Socket. The opened Socket is only valid for one thread. Another thread may legally try to OPEN the ServerSocket again, thus giving another standard Socket. This allows each thread to service a client and makes the ServerSocket concept meaningful. All operations other than OPEN are actually performed on the client connected socket. If a socket connection is broken, it may be detected as an INVALID KEY condition in a READ/WRITE verb.
Socket:host:port	Sequential Stream Only. A TCP/IP Socket is created on the given port. If a socket connection is broken, it may be detected as an INVALID KEY condition in a READ/WRITE verb.
Ssliteclient:	Secure Socket Layer version of socket: using IBM's SSLite.
Ssliteserver:	Secure Socket Layer version of server: using IBM's SSLite.
Syserr:	Output only. System error, suitable for command-line redirection. This is the same device used for DISPLAY UPON SYSERR.
System:filename	System console, suitable for command-line redirection, this is output for the system console, system terminal, command prompt, etc. This is the same device used for DISPLAY UPON SYSOUT, ACCEPT FROM SYSIN.
Sys:filename	Synonym for system
Sysout:filename	Synonym for system
Sysin:filename	Synonym for system
Stdout:filename	Synonym for system
Stdin:filename	Synonym for system
Url:url-filename	Sequential Stream Only. The url-filename is used as a generalized URL. This allows any URL known to Java to function. No special checking of the validity of the URL is done. If running as an applet, any remote host names in the URL must be the webserver.
Documentbase:filename	Filename treated as http: reference starting at an applet's document base
Codebase:filename	Filename treated as http: reference starting at an applet's code base.
Loaddialog:dialog_filename/title_of_dialog/file_filter/file_directory	Prompt user with Open dialog; filename given by user is then used.
Savedialog:dialog_filename/title_of_dialog/file_filter/file_directory	Prompt user with Save dialog; filename given by user is then used.
Ro:filename	Treat file as Read Only; filename is processed further.
Wo:filename	Treat file as Write Only; filename is processed further.
line:filename	The file will be treated as LINE SEQUENTIAL; filename is processed further.
raw:filename	Sequential Only. Where appropriate, file access is processed as raw data rather than cooked data with header information. Filename is processed further.
env:environment	The configuration parameter 'environment' is retrieved and used as the filename for further processing.
<filename	Synonym for ipcio:filename
>filename	Synonym for ipcio:filename

|filename	Synonym for ipcio:filename
-p filename	Synonym for ipcio:filename
-P filename	Synonym for ipcio:filename
-f filename	Synonym for filename
-F filename	Synonym for filename

9. Compilation group structure

A compilation group is a series of source units. A source unit may contain other source units and these contained source units may reference some of the resources of the source units within which they are contained.

Organization

A source unit begins with an identification division and includes any contained source units.

With the exception of compiler directives, source text manipulation statements, and end markers, the statements, entries, paragraphs, and sections of a source unit are grouped into four divisions that are sequenced in the following order:

1. identification division
2. environment division
3. data division
4. procedure division

The beginning of a division in a source unit is indicated by the appropriate division header. The beginning of the identification division may also be indicated by one of the paragraph headers permitted in the identification division.

The end of a division in a source unit is indicated by one of the following:

1. The beginning of a succeeding division in that source unit
2. The end marker for that source unit
3. That physical position after which no more source lines occur.

The end of a source unit is indicated by an end marker, if specified, or by the absence of additional source lines in the compilation group.

A source unit that is directly or indirectly contained within another source unit is considered in these specifications as a separate source unit that may additionally reference certain resources defined in the containing source unit.

The object code resulting from compiling a source unit contained within another source unit is considered in these specifications to be inseparable from the object code resulting from compiling the containing source unit.

A source element is a source unit excluding any nested source units.

A run time entity is the result of compiling a source element.

COBOL compilation group

General format

{{ program-definition } | class-definition}…

where program-definition is:

[{IDENTIFICATION | ID} DIVISION.]
PROGRAM-ID.
{program-name-1 | literal-1 | program-name-1 AS literal-1} [INHERITS class-literal]
[IS [[NOT] COMMON] [[NOT] INITIAL] [[NOT] RESIDENT] PROGRAM].
{PROGRAM-ID|PROGRAM|PROGRAM IDENTIFICATION}.
{program-name-1|literal-1} [AS literal-1][INHERITS literal-2].
{{IS [NOT] COMMON} | {IS [NOT] RESIDENT} | {IS [NOT] INITIAL}|
{IS [NOT] EXTERNAL} | {IS [NOT] RECURSIVE} | {IS [NOT] FINAL}|
{IS [NOT] PUBLIC} | {IS PRIVATE} | {IS PROTECTED} | {IS PACKAGE}} PROGRAM
…
[IMPLEMENTS literal-3…]
[environment-division]
[data-division]
[procedure-division [program-definition] …]
[END PROGRAM program-name-1 | literal-1.]
.

Where class-definition is:

[{IDENTIFICATION | ID} DIVISION.]
CLASS-ID.
{program-name-1 | literal-1 | program-name-1 AS literal-1} [INHERITS class-literal]
[IS [[NOT] COMMON] [[NOT] INITIAL] [[NOT] RESIDENT] PROGRAM].
{PROGRAM-ID|PROGRAM|PROGRAM IDENTIFICATION}.
{program-name-1|literal-1} [AS literal-1][INHERITS literal-2].
{{IS [NOT] COMMON} | {IS [NOT] RESIDENT} | {IS [NOT] INITIAL}|
{IS [NOT] EXTERNAL} | {IS [NOT] RECURSIVE} | {IS [NOT] FINAL}|
{IS [NOT] PUBLIC} | {IS PRIVATE} | {IS PROTECTED} | {IS PACKAGE}} PROGRAM
…
[IMPLEMENTS literal-3…]
.
[environment-division]
[data-division]
[procedure-division [program-definition] …]
[method-definition]…
END CLASS program-name-1 | literal-1

where method-definition is:

[{IDENTIFICATION | ID} DIVISION.]
METHOD-ID.
{program-name-1 | literal-1 [METHOD OVERRIDE]

{IS [NOT] COMMON} through {IS PACKAGE} PROGRAM from above
[environment-division]
[data-division]
[procedure-division]
END METHOD program-name-1 | literal-1.
[END PROGRAM program-name-1 | literal-1.]

Syntax rules

1. There are three types of COBOL source units: PROGRAM, CLASS and METHOD.
2. CLASS units must contain only METHOD units.
3. METHOD units must be contained by CLASS units.
4. METHOD OVERRIDE indicates that the method definition overrides a definition in a superclass; this is only commentary.
5. CLASS units INHERIT (also known as extend) a superclass (also known as a parent or base class).
6. In Elastic COBOL all CLASS units, even those without an INHERITS clause, eventually inherit from java.lang.Object.
7. In Elastic COBOL, PROGRAM units are implicitly Java Class objects.
8. PROGRAM is the traditional COBOL source unit.
9. CLASS units may be instantiated (created) by either Elastic COBOL programs or by Java programs.
10. All top-level source unites names are produced in lowercase with dashes (-) converted to underscores(_) if not specified as a literal.
11. An end marker shall be present in every source unit that contains, is contained in, or precedes another source unit.
12. If program-name-1 and literal-1 are specified, literal-1 is the name used.
13. Class-literal is the literal name of a class, e.g., “java.applet.Applet”. This is optional and should not be changed unless the program should inherit from other than the default, “java.applet.Applet”. A repository classname may not be specified here.
14. The RESIDENT clause disables the CANCEL verb’s effects on this program.
15. Program-name-1 shall be identical to the program-name declared in a preceding PROGRAM-ID paragraph.
16. If a PROGRAM-ID paragraph declaring a specific program-name is stated between the PROGRAM-ID paragraph and the END PROGRAM header for program-name-1, then an END PROGRAM header referencing program-name shall precede the END PROGRAM header referencing program-name-1.
17. COMMON on a nested PROGRAM source unites indicates that the program many be called by other programs.
18. INITIAL indicates that the program's state should be reinitialized with each CALL.

19. RECURSIVE indicates that a program may CALL itself. This is treated as commentary because all Elastic COBOL program may be called recursively.
20. FINAL indicates that this unit may not be inherited or overridden.
21. PUBLIC indicates that any program may CALL or INVOKE this program. This is the default.
22. PRIVATE or NOT PUBLIC indicates that only programs in the overall source unit may CALL or INVOKE this program. If a METHOD, only other METHODs in the same CLASS may invoke it.
23. PROTECTED indicates that only programs in this overall source unit or in source units extending this source unit may CALL or INVOKE this program.

End markers

End markers indicate the end of a definition.

General format

END PROGRAM {program-name-1 | literal-1}.

General rules

1. An end marker indicates the end of the specified source unit.
2. If the source unit terminated by the end marker is contained within another source unit, the next statement shall be either the first statement of a source unit or another end marker that terminates the containing source unit.
3. If the program terminated by an end marker is not contained within another source unit, the next statement shall be the first statement of a source unit to be compiled separately from the source unit terminated by the end marker.

External repository

The external repository stores information specified in program definitions and class definitions.

The information stored about these source units must consist of all information required for activation and checking conformance. This information includes

- the externalized name of the source unit
- the type of the source unit - program, function, class, or interface
- the parameters of the source unit, if any
- the returning item of the source unit, if any
- the call convention of the source unit, if any
- the object properties of the source unit, if any
- the methods contained in the source unit, if any, and details about the method's externalized name, parameters, returning item, and call convention

- type declarations required for the description of parameters and returning items
- whether the DECIMAL-POINT IS COMMA clause is specified in the source unit;
- whether the CURRENCY clause is specified in the source unit; This information about a source unit, excluding the externalized name of the source unit, is called its signature.

The details on the association of the name of a source unit with information in the external repository are specified in REPOSITORY paragraph.

Program organization and communication

A compilation group may contain zero, one, or more source units. A source unit may contain other source units, and these contained source units may reference some of the resources of the source unit in which they are contained. (See COBOL compilation group, for full details of the structure.)

When a source unit, B, is contained in another source unit, A, it may be directly or indirectly contained. Source unit B is directly contained in source unit A if there is no source unit contained in A that also contains B. Source unit B is indirectly contained in source unit A if there exists a source unit contained in A that also contains B.

Objects and classes

An object is a single entity consisting of data and methods. An object belongs to a class. A class describes the structure of the data and the methods that apply to all the objects belonging to that class. A class has any number of constructors that are used to construct the object from the class. Elastic COBOL classes automatically include a default constructor.

Object references

An object reference is a value that uniquely identifies an object for the lifetime of the object. No two distinct objects have the same object reference and every object has at least one object reference.

It is permitted to have more than one object reference for any given object provided the requirements of this standard are fully met, but it is sufficient that a given object has precisely one object reference.

In Elastic COBOL, object references are outside the scope of traditional COBOL memory. In all COBOL implementations with object references, the references are transitory and invalid beyond the specific run. Object references in Elastic COBOL are direct Java object references and as such possess all the capabilities and restrictions inherent in Java objects, including visibility of methods, security, etc. An object is garbage collected at any point after there are no longer any references to it. No explicit free is required.

Methods

The procedural code in an object is placed in methods. Each method has its own method-name and its own data division and procedure division. When a method is invoked, the procedural code it contains is executed. A method is invoked by specifying an identifier that references the object and the name of the method. A method may specify parameters and a returning item.

File connector

A file connector is a storage area that contains information about a file and is used as the linkage between a file-name and a physical file. A file connector is either internal or external as described in External and internal items.

A file connector is placed in an open mode by the execution of a successful OPEN statement that references the associated file-name. The OPEN statement also associates the file connector with a physical file. When a CLOSE statement references the associated file-name, the file connector is no longer associated with the physical file and the file connector is no longer in an open mode. In the following cases, the COBOL runtime system executes an implicit CLOSE statement without any optional phrases for a file connector that is in the open mode:

- When the run unit terminates.
- For initial file connectors described in a program when a GOBACK or an EXIT PROGRAM statement is executed in a called program in which they are described.
- For file connectors in the program to which a CANCEL statement is executed or in any program contained in that program.
- For file connectors in an object when the object is deleted.

Global names and local names

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

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

A file-name, record-name, or report-name described using a GLOBAL clause is a global name. All data-names subordinate to a global name are global names. All condition-names associated with a global name are global names.

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

A data-name or file-name declared in a source element for an object definition or a factory definition is global.

Global names are transitive across source elements contained within other source elements.

External and internal items

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 it is declared.

A data item or file connector is external if the storage associated with that item is associated with the run unit rather than with any particular program within the run unit. An external item may be referenced by any program in the run unit that describes it. References to external items from different programs using separate descriptions of the data item or file connector are always references to the same item. In a run unit, there is only one representation of an external item.

A data item or file connector is internal if the storage associated with it is associated only with the program that describes it.

External and internal data items and file connectors 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 also attains the external attribute. If a record or data item does not have the external attribute, it is part of the internal data of the program in which it is described.

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

Data records, subordinate data items, and various associated control information described in the communication, linkage, local-storage, and report sections of a program are always considered to be internal to the program describing that data. Special considerations apply to data described in the linkage section whereby an association is made between the data records described and other data items accessible to other programs.

Automatic, initial, and static items

There are three kinds of internal data items and file connectors: automatic, initial, and static. The designation of automatic, initial, and static items relates to their persistence and the persistence of their contents during the execution of a run unit.

Data items and file connectors have an initial and last-used state. The initial state of a data item depends on the presence or absence of a VALUE clause in its data description entry. If a VALUE clause is present, when a data item defined in the working-storage or local-storage section of a program is set to its initial state, it is set to the associated value. If a data item is defined in any other section or its data description entry does not include a VALUE clause, its content is undefined when it is set to its initial state. The initial state of a file connector is that it is not in an open mode.

Elastic COBOL uses the VALUE clause for initial definition regardless of section.

Last-used state means that the content of the data item or file connector is that of the last time it was modified.

Automatic items are set to their initial state any time a function, method, or program is activated, and each instance of the function, method, or program has its own copy of the item. An automatic item is an item described in the local-storage section

Initial items are set to their initial state any time an initial program is activated. All data items and file connectors in an initial program are initial items.

Static items are set to their initial state any time a method, or program is set to its initial state. (See State of a method, object, or program.) A static item is an item described in the communication section or in the file or working-storage section of a source unit that is not an initial program.

Common, initial, and resident programs

Programs that form part of a run unit may possess zero, one, or more of the following attributes: common, initial, and resident.

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

An initial program is one whose program state is initialized when the program is called. During the process of initializing an initial program, that program's internal data is initialized as described in State of a function, method, object, or program. The initial attribute is attained by specifying the INITIAL clause in the program's identification division.

A resident program is one upon which the CANCEL verb has no effect; no re-initialization code is generated for the CANCEL. If the CANCEL verb is not to be used with the program, RESIDENT programs have a smaller code-size.

Sharing data

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

1. The data content of an external data record may be referenced from any run time entity provided that run time entity has described that data record.
2. If a program is contained within another program, both programs may refer to data possessing the global attribute either in the containing program or in any program that directly or indirectly contains the containing program.
3. The mechanism whereby an argument value is passed by reference from an activating run time entity to an activated run time entity establishes a common data item. The activated entity and the activating entity may use a different name to refer to the common data item.

Sharing file connectors

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

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

Method invocation

The procedural code in a method is executed by invoking the method with an INVOKE statement. The method implementation that is bound to the invocation depends on the class, at run time, of the object on which the method is invoked. In particular, it is not the class specified statically on the definition of the object reference identifier, it is the class of the object that is referenced at run time that is used in resolving a method invocation to a particular method implementation.

If SUPER is specified as the object identifier, then the invocation will resolve using a restricted search, as specified in SELF and SUPER.

Method resolution proceeds by applying the first one of the following rules that is applicable:

1. If a method with the method-name specified in the invocation is defined in the class of the object, that method is bound; otherwise,
2. If a method with the method-name specified in the invocation is defined in one of the classes that is inherited by the class of the object, that method is bound. Inherited classes are inspected in turn including any classes inherited from higher levels of the class hierarchy.

Program results

The contents of the RETURN-VALUE special register are used as a program's return result when returning to the operating system.

Class inheritance

Class inheritance is a mechanism for using the interface and implementation of one or more classes as the basis for another class. The inheriting class, also known as a subclass, inherits from one class, known as the superclass. The subclass has all the methods defined for the inherited class definition, including any methods that the inherited definition or definitions inherited. The subclass has all the data definitions defined for the inherited class or classes, including any data definitions that the inherited class or classes inherited. These inherited data definitions define inherited data for every object of the subclass and for its factory. The inherited object data are initialized when an object is created. The inherited factory data are allocated independently from the factory data of the inherited class or classes and are initialized when the factory of the subclass is created. The inherited factory data are visible only to factory methods defined in the class that declared the data. The inherited object data are visible only to object methods defined in the class that declared the data. The subclass may define new methods and additional data augmenting the set of inherited methods and the inherited data.

Elastic COBOL and Java support single inheritance; that is, inheriting from only one class. Elastic COBOL programs can currently inherit only from one Java class using the INHERITS clause in the identification division. By default, Elastic COBOL programs inherit from java.applet.Applet. Only change this if necessary for the program; if not inheriting from java.applet.Applet or a subclass of java.applet.Applet, the Elastic COBOL program cannot be run as an applet. All Java objects, including Elastic COBOL programs, ultimately inherit from java.lang.Object, and include its methods.

Object life cycle

The life cycle for an object begins when it is created and ends when it is garbage collected.

Garbage collection is the process of automatically removing from memory those objects which can no longer be accessed from any live thread of execution. This occurs transparently to the user and programmer and is the responsibility of the Java Virtual Machine.

Life cycle for objects

An object is created when its constructor is called. The constructor is an unnamed method automatically called when no method name is given in an INVOKE.

An object is destroyed either when it is determined that the object cannot take part in the continued execution of the run unit, or when the run unit terminates, whichever occurs first.

The timing and algorithm for the mechanism that determines whether or not an object can take part in the continued execution of the run unit is defined by the Java implementation.

Note -- Java implementations vary in the method used to collect the objects, but they fall into two broad categories – conservative and exact. Conservative collectors can only make educated guesses about when an object can no longer be referenced, and as such, some garbage may be left in memory until program termination. Exact collectors, as the name implies, are exact about which objects may or may not be referenced and are more stable for long-running programs. If the same run unit executes for a long period of time, use a Java implementation with an exact collector. Other classifications, such as generational, copying, etc. refer to the algorithms used in the collector, particularly the efficiency, and do not affect stability; they may affect speed and scalability across multiple processors. Different Java implementations are better for different programs; before deploying, it can help to try your program with different Java implementations to see which is best for the program in question.

10. Identification Division

The identification division identifies the program, class or method.

The paragraph header identifies the type of information contained in the paragraph.

General format

[IDENTIFICATION DIVISION.]

{{program-id-paragraph | {class-id-paragraph} | {method-id-paragraph}}

Program definition

A program definition is a source unit introduced by an identification division containing the PROGRAM-ID paragraph.

A class definition is a source unit introduced by an identification division containing the CLASS-ID paragraph.

A method definition is a source unit introduced by an identification division containing the METHOD-ID paragraph.

A program definition may contain an environment division, a data definition and a procedure division. A program definition may end with an END PROGRAM header.

A class definition may contain an environment division, a data division and a procedure division. A class definition MUST end with an END CLASS header.

A method definition may contain an environment division, a data division and a procedure division. A method definition MUST end with an END METHOD header. A method definition must be contained within a class definition.

PROGRAM-ID Paragraph

The PROGRAM-ID paragraph specifies the name by which a program is identified and assigns selected program attributes to that program.

General format

{PROGRAM-ID|PROGRAM|PROGRAM IDENTIFICATION}.
{program-name-1|literal-1} [AS literal-1][INHERITS literal-2].
{{IS [NOT] COMMON}|
{IS [NOT] RESIDENT}|
{IS [NOT] INITIAL}|
{IS [NOT] EXTERNAL}|
{IS [NOT] RECURSIVE}|
{IS [NOT] FINAL}|
{IS [NOT] PUBLIC}|
{IS PRIVATE}|
{IS PROTECTED}|
{IS PACKAGE}} PROGRAM …

[IMPLEMENTS literal-3…]

.

Syntax rules

PROGRAM is the traditional COBOL source unit.

Literal-1 shall be an alphanumeric literal or a national literal and shall not be a figurative constant.

If both program-name-1 and literal-1 are specified by using AS keyword, literal-1 takes precedence for naming.

If specified, literal-1 is the Java class name. If program-name-1 is used, all letters are converted to lowercase, and all dashes are converted to underscores in the Java class name. If the program being specified is a nested COBOL program, the Java class name is preceded by the containing program's class name followed by a dollar sign '$' followed by the Java class name of the subprogram.

A program contained within another program shall not be assigned the same name as that of any other program contained within the separately compiled program that contains this program.

The optional COMMON clause may only be used if the program is contained within another program.

The INHERIT clause specifies from which class this program should inherit functionality.

General rules

Literal-1, if specified, is the name of the program that is externalized to the operating environment.

The COMMON clause specifies that the program is common. A common program is contained within another program, but may be called from programs other than the program containing it. (See Scope of names.)

The INITIAL clause specifies that the program is initial. When an initial program is activated, the data items and file connectors contained in it and any program contained within it are set to their initial states.

The RESIDENT clause specifies that the program is resident. When another program attempts to CANCEL a resident program, the cancel is ineffective.

The INHERITS clause specifies that the program should inherit functionality from the named class. By default, Elastic COBOL programs inherit from java.applet.Applet; changing this will make Elastic COBOL programs no longer functional as applets unless the literal-2 inherits in turn from Applet (such as Japplet). All Java classes, including Elastic COBOL programs, inherit ultimately from java.lang.Object and can inherit only from one superclass (single-inheritance).

The RECURSIVE attribute indicates that a program may CALL itself. This is treated as commentary because all Elastic COBOL programs may be called recursively.

The FINAL attribute indicates that this unit may not be inherited or overridden.

The PUBLIC attribute indicates that any program may CALL or INVOKE this unit. This is the default.

The PRIVATE or NOT PUBLIC attribute indicates that only programs in this overall source unit may CALL or INVOKE this unit.

The PROTECTED attribute indicates that only programs in this overall source unit or in source units extending this source unit may CALL or INVOKE this unit.

CLASS-ID Paragraph

The CLASS-ID paragraph specifies the name by which a class is identified and assigns selected program attributes to that class.

General format

{CLASS-ID}.
{program-name-1|literal-1} [AS literal-1][INHERITS literal-2].
{{IS [NOT] COMMON}|
{IS [NOT] RESIDENT}|
{IS [NOT] INITIAL}|
{IS [NOT] EXTERNAL}|
{IS [NOT] RECURSIVE}|
{IS [NOT] FINAL}|
{IS [NOT] PUBLIC}|
{IS PRIVATE}|
{IS PROTECTED}|
{IS PACKAGE}} PROGRAM …
[IMPLEMENTS literal-3…]
.

Syntax rules

Literal-1 shall be an alphanumeric literal or a national literal and shall not be a figurative constant.

If both program-name-1 and literal-1 are specified by using AS keyword, literal-1 takes precedence for naming.

If specified, literal-1 is the Java class name. If program-name-1 is used, all letters are converted to lowercase, and all dashes are converted to underscores in the Java class name. If the program being specified is a nested COBOL program, the Java class name is preceded by the containing program's class name followed by a dollar sign '$' followed by the Java class name of the subprogram.

A program contained within another program shall not be assigned the same name as that of any other program contained within the separately compiled program that contains this program.

The optional COMMON clause may only be used if the program is contained within another program.

CLASS units INHERIT (also known as extend) exactly one superclass (also known as a parent or base class).

The INHERIT clause specifies from which class this program should inherit functionality.

CLASS units may IMPLEMENT multiple interfaces. An interface is not a class, but defines the structure which a class must have. Note that currently java.lang.Runnable and java.io.Serializable may not be specified as all Elastic COBOL programs implement these interfaces already.

General rules

Literal-1, if specified, is the name of the program that is externalized to the operating environment.

The COMMON clause specifies that the program is common. A common program is contained within another program, but may be called from programs other than the program containing it. (See Scope of names.)

The INITIAL clause specifies that the program is initial. When an initial program is activated, the data items and file connectors contained in it and any program contained within it are set to their initial states.

The RESIDENT clause specifies that the program is resident. When another program attempts to CANCEL a resident program, the cancel is ineffective.

The INHERITS clause specifies that the program should inherit functionality from the named class. By default, Elastic COBOL programs inherit from java.applet.Applet; changing this will make Elastic COBOL programs no longer functional as applets unless the literal-2 inherits in turn from Applet (such as Japplet). All Java classes, including Elastic COBOL programs, inherit ultimately from java.lang.Object and can inherit only from one superclass (single-inheritance).

CLASS units may be instantiated (created) by either Elastic COBOL programs or Java programs.

The RECURSIVE attribute indicates that a program may CALL itself. This is treated as commentary because all Elastic COBOL programs may be called recursively.

The FINAL attribute indicates that this unit may not be inherited or overridden.

The PUBLIC attribute indicates that any program may CALL or INVOKE this unit. This is the default.

The PRIVATE or NOT PUBLIC attribute indicates that only programs in this overall source unit may CALL or INVOKE this unit.

The PROTECTED attribute indicates that only programs in this overall source unit or in source units extending this source unit may CALL or INVOKE this unit.

METHOD-ID Paragraph

The METHOD-ID paragraph specifies the name by which a method is identified and assigns selected program attributes to that method.

General format

{METHOD-ID}.

{program-name-1|literal-1} [AS literal-1][METHOD OVERRIDE].

{{IS [NOT] COMMON}|

{IS [NOT] RESIDENT}|

{IS [NOT] INITIAL}|

{IS [NOT] EXTERNAL}|
{IS [NOT] RECURSIVE}|
{IS [NOT] FINAL}|
{IS [NOT] PUBLIC}|
{IS PRIVATE}|
{IS PROTECTED}|
{IS PACKAGE}} PROGRAM ...

.

Syntax rules

Literal-1 shall be an alphanumeric literal or a national literal and shall not be a figurative constant.

If both program-name-1 and literal-1 are specified by using AS keyword, literal-1 takes precedence for naming.

If specified, literal-1 is the Java class name. If program-name-1 is used, all letters are converted to lowercase, and all dashes are converted to underscores in the Java class name. If the program being specified is a nested COBOL program, the Java class name is preceded by the containing program's class name followed by a dollar sign '$' followed by the Java class name of the subprogram.

A program contained within another program shall not be assigned the same name as that of any other program contained within the separately compiled program that contains this program.

The optional COMMON clause may only be used if the program is contained within another program.

The INHERIT clause specifies from which class this program should inherit functionality.

METHOD OVERRIDE indicates that the method definition overrides a definition in a superclass. This clause is only for commentary.

General rules

Literal-1, if specified, is the name of the program that is externalized to the operating environment.

The COMMON clause specifies that the program is common. A common program is contained within another program, but may be called from programs other than the program containing it. (See Scope of names.)

The INITIAL clause specifies that the program is initial. When an initial program is activated, the data items and file connectors contained in it and any program contained within it are set to their initial states.

The RESIDENT clause specifies that the program is resident. When another program attempts to CANCEL a resident program, the cancel is ineffective.

The INHERITS clause specifies that the program should inherit functionality from the named class. By default, Elastic COBOL programs inherit from java.applet.Applet; changing this will make Elastic COBOL programs no longer functional as applets unless the literal-2 inherits in turn from Applet (such as Japplet). All Java classes,

including Elastic COBOL programs, inherit ultimately from java.lang.Object and can inherit only from one superclass (single-inheritance).

The RECURSIVE attribute indicates that a program may CALL itself. This is treated as commentary because all Elastic COBOL programs may be called recursively.

The FINAL attribute indicates that this unit may not be inherited or overridden.

The PUBLIC attribute indicates that any program may CALL or INVOKE this unit. This is the default.

The PRIVATE or NOT PUBLIC attribute indicates that only programs in this overall source unit may CALL or INVOKE this unit. Only other METHODs in this CLASS may INVOKE this unit.

The PROTECTED attribute indicates that only programs in this overall source unit or in source units extending this source unit may CALL or INVOKE this unit.

11. Environment Division

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

General format

[ENVIRONMENT DIVISION.]

[configuration-section]

[input-output-section]

Configuration Section

The configuration section specifies aspects of the data processing system that are dependent on the specific system as well as special control techniques and a means of associating a local name with an external resource. This section is divided into paragraphs.

The SOURCE-COMPUTER paragraph, which describes the computer configuration on which the source element is compiled.

The OBJECT-COMPUTER paragraph, which describes the computer configuration on which the object program produced by the compiler is to be run.

The SPECIAL-NAMES paragraph, which provides a means for specifying the currency sign, selecting the decimal point, specifying symbolic-characters, relating implementor-names to user-specified mnemonic-names, relating alphabet-names to character sets or collating sequences, and relating class-names to sets of characters.

The REPOSITORY paragraph, which provides a means for associating a local name with an external resource.

The FIGURATIVE-CONSTANTS paragraph associate a figurative constant with a value for Wang compatibility.

General format

CONFIGURATION SECTION.

[source-computer-paragraph]

[object-computer-paragraph]

[special-names-paragraph]

[repository-paragraph]

[figurative-constants-paragraph]

Syntax rules

1. The configuration section shall not be specified in a program that is contained within another program.

2. The configuration section may be specified in a program definition..

General rules

The entries explicitly or implicitly specified in the configuration section of a program that contains other programs apply to each contained program.

SOURCE-COMPUTER Paragraph

The SOURCE-COMPUTER paragraph provides a means of describing the computer upon which the program is to be compiled.

General format

SOURCE-COMPUTER. [computer-name [WITH DEBUGGING MODE].]

General rules

1. All clauses of the SOURCE-COMPUTER paragraph apply to the program in which they are explicitly or implicitly specified and to any program contained within that program.
2. When the SOURCE-COMPUTER paragraph is not specified and the program is not contained within a program including a SOURCE-COMPUTER paragraph, the computer on which the source program is being compiled is the source computer.
3. When the SOURCE-COMPUTER paragraph is specified, but the computer-name is not specified, the computer upon which the source program is being compiled is the source computer.
4. If the WITH DEBUGGING MODE clause is specified in a program, all debugging lines are compiled as if the indicator area contained a space.
   The WITH DEBUGGING MODE also implies inclusion of debugging information sufficient for the Elastic COBOL debugger.

   NOTE - The DEBUGGING MODE clause is an obsolete element in this draft International Standard and is to be deleted from the next revision of standard COBOL.
5. If the WITH DEBUGGING MODE clause is not specified in a program and the program is not contained within a program including a WITH DEBUGGING MODE clause, any debugging lines are compiled as if they were comment lines.
6. The -debug compiler option implies WITH DEBUGGING MODE.

OBJECT-COMPUTER Paragraph

The OBJECT-COMPUTER paragraph provides a means of describing the computer on which the program is to be executed.

General format

OBJECT-COMPUTER.

[computer-name

[MEMORY SIZE integer-1 {WORDS|CHARACTERS|MODULES}

[PROGRAM COLLATING SEQUENCE IS alphabet-name-1]
[SEGMENT-LIMIT integer-2]

.]

Syntax rules

Alphabet-name-1 shall reference an alphabet that defines an alphanumeric collating sequence.

General rules

1. The computer-name may provide a means for identifying equipment configuration. If computer-name is WANG, certain warnings are disabled.
2. All clauses of the OBJECT-COMPUTER paragraph apply to the program in which they are explicitly or implicitly specified and to any program contained within that program.
3. When the OBJECT-COMPUTER paragraph is not specified and the program is not contained within a program including an OBJECT-COMPUTER paragraph, the object computer is JVM.
4. When the OBJECT-COMPUTER paragraph is specified, but the computer-name is not specified, the object computer is JVM.
5. The memory size clause is treated as documentary only.
6. When the PROGRAM COLLATING SEQUENCE clause is specified, the initial alphanumeric program collating sequence is the collating sequence associated with alphabet-name-1. When alphabet-name-1 is not specified, the initial alphanumeric program collating sequence is the native alphanumeric collating sequence.
7. When the PROGRAM COLLATING SEQUENCE clause is not specified and the program is not contained within a program for which a PROGRAM COLLATING SEQUENCE clause is specified, the initial program collating sequences are the native alphanumeric collating sequence and the native national collating sequence.
8. The alphanumeric program collating is used to determine the truth value of any alphanumeric comparisons and national comparisons, respectively, that are:
    a. Explicitly specified in relation conditions.
    b. Explicitly specified in condition-name conditions.
9. The alphanumeric program collating sequence explicitly or implicitly established by the OBJECT-COMPUTER paragraph is effective with the initial state of the programs to which they apply.
10. The alphanumeric program collating sequence and national program collating sequence are applied to alphanumeric and national sort or merge keys, respectively, unless the sort or merge collating sequence has been modified by execution of a SET statement or a COLLATING SEQUENCE phrase is specified in the respective SORT or MERGE statement.

The SEGMENT-LIMIT clause is treated as documentary only.SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph provides a mechanism for defining platform, system and vendor specific information.

General format

SPECIAL-NAMES.
[switch-clause]…
[implementor-clause]…
[alphabet-clause]…
[symbolic-clause]…
[class-clause]…
[currency-clause]…
[decimal-point-clause]…
[cursor-clause]…
[crt-status-clause]…
[screen-control-clause]…
[event-status-clause]…
[call-convention-clause]…
[return-code-clause]…
[dynamic-configuration-clause]…
[dynamic-environment-clause]…
[linkage-clause]…

The clause in SPECIAL-NAMES may be specified in any order subject to their using items defined elsewhere in the SPECIAL-NAMES; if using items defined elsewhere in the SPECIAL-NAMES, the dependency must be defined first.

General rules

1. All clauses specified in the SPECIAL-NAMES paragraph for a program also apply to programs contained within that program. The condition-names specified in the containing program's SPECIAL-NAMES paragraph may be referenced from any contained program.
2. Switch-name-1 identifies a switch from SWITCH-1 through SWITCH-26. The on status and/or off status of an external switch may be associated with condition-names. The status of that switch may be interrogated by testing these condition-names (see Switch-status condition).
3. The status of a switch may be altered by execution of a SET mnemonic-name statement that specifies as its operand the mnemonic-name associated with that switch.
4. The NUMERIC SIGN clause sets the default numeric storage for the program.
5. The FILE CHARACTER sets the implicit directory separator character to literal-8, e.g., “/” IS FILE CHARACTER or “\” IS FILE CHARACTER.

6. The INITIAL LINKAGE loads the library (.dll, .so, etc.) literal-9 at runtime. Do not specify the .dll or .so, or the initial 'lib' in Posix.
7. The INITIAL CLASS loads the class literal-10 at runtime (Java initial class libraries such as JDBC drivers). Do not specify the .class ending.
8. The ALPHABET clause provides a means of relating a name to a specified coded character set and collating sequence.

   NOTE - An alphabet defines both a coded character set and a collating sequence. An alphabet-name referenced in the PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph or in the COLLATING SEQUENCE phrase of a SORT, MERGE, or SET statement references a collating sequence. An alphabet-name referenced in a SYMBOLIC CHARACTERS clause or in the CODE-SET clause of a file description entry references a coded character set.

   When the ALPHABET clause is specified:

   a. STANDARD-1 and STANDARD-2 in Elastic COBOL refer to the Unicode character set and the ASCII equivalent characters which begin the Unicode character set.
   b. When the NATIVE phrase is specified, the native alphanumeric coded character set and native alphanumeric collating sequence are referenced; otherwise, the native national coded character set and native national collating sequence are referenced.
   c. When literal-phrase is specified, the coded character set and/or collating sequence is defined according to the following rules, where the native coded character set is the type of coded character set or collating sequence being defined, either alphanumeric or national:

      - The value of each literal specifies:
      - The ordinal number of a character within the native character set, if the literal is numeric. This value shall not exceed the value that represents the number of characters in the native character set.
      - Otherwise, the actual character within the native character set. If the value of the literal contains multiple characters, each character in the literal, starting with the leftmost character, is assigned successive ascending positions in the collating sequence being specified.
      - The order in which the literals appear in the ALPHABET clause specifies, in ascending sequence, the ordinal number of the character within the collating sequence being specified.
      - Any characters of the native collating sequence that are not specified in the literal phrase shall assume a position in the collating sequence that is greater than that of the highest character specified in this literal phrase. The relative order within the set of these unspecified characters is unchanged from the native collating sequence.
      - If a character code set is being specified, the implementor defines the ordinal number within the character code set being specified for each

character within the native character set that is not specified by the literal-1 phrase.

- If the THROUGH phrase is specified, the set of contiguous characters in the native character set beginning with the character specified by the value of literal-1, and ending with the character specified by the value of literal-2, is assigned a successive ascending position in the collating sequence being specified. In addition, the set of contiguous characters specified by a given THROUGH phrase may specify characters of the native character set in either ascending or descending sequence.

- If the ALSO phrase is specified, the characters of the native character set specified by the value of literal-1 and literal-3 are assigned to the same ordinal position in the collating sequence being specified or in the character code set that is used to represent the data, and if alphabet-name-1 is referenced in a SYMBOLIC CHARACTERS clause, only literal-1 is used to represent the character in the native character set.

9. The character that has the highest ordinal position in the program collating sequence is associated with the figurative constant HIGH-VALUE, except when this figurative constant is specified as a literal in the SPECIAL-NAMES paragraph. If more than one character has the highest position in the program collating sequence, the last character specified is associated with the figurative constant HIGH-VALUE.

10. The character that has the lowest ordinal position in the program collating sequence is associated with the figurative constant LOW-VALUE, except when this figurative constant is specified as a literal in the SPECIAL-NAMES paragraph. If more than one character has the lowest position in the program collating sequence, the first character specified is associated with the figurative constant LOW-VALUE.

11. When specified as literals in the SPECIAL-NAMES paragraph, the figurative constants HIGH-VALUE and LOW-VALUE are associated with those characters having the highest and lowest positions, respectively, in the native alphanumeric collating sequence otherwise.

12. When the SYMBOLIC CHARACTERS clause is specified:

    a. Symbolic-character-1 defines a figurative constant.

    b. The value of figurative constant symbolic-character-1 is the internal representation of the character at ordinal position integer-1 in the native alphanumeric character set or, if the IN phrase is specified, in the character set referenced by alphabet-name-3.

13. The CLASS clause provides a means for relating a name to the specified set of characters listed in that clause. The characters specified by the values of the literals in this clause define the exclusive set of characters of which class-name-1 consists.

    The value of each literal specifies:

a. When the literal is numeric, the ordinal number of a character within the relevant native character set.

b. Otherwise, the actual character within the relevant native character set. If the value of literal-5 contains multiple characters, each character in the literal is included in the set of characters identified by class-name-1.

If the THROUGH phrase is specified, the contiguous characters in the native character set beginning with the character specified by the value of literal-5, and ending with the character specified by the value of literal-6, are included in the set of characters identified by class-name-1. In addition, the contiguous characters specified by a given THROUGH phrase may specify characters of the native character set in either ascending or descending sequence.

14. The CURRENCY SIGN clause is used to specify a currency string that is placed into numeric-edited data items when they are used as receiving items and de-edited from the data item when it is used as a sending item that has a numeric or numeric-edited receiving item. In addition, it is used to determine which symbol shall be used in a PICTURE character string to specify the presence of this currency string. This symbol is referred to as the currency symbol.

    Literal-7 represents the value of the currency string.

15. The clause DECIMAL-POINT IS COMMA means that the functions of comma and period are interchanged in the character-string of the PICTURE clause and in numeric literals.

16. The content of the data item referenced by data-name-2 specifies the position of the cursor at the beginning of the execution of an ACCEPT screen statement. This content shall be updated by the execution of a successful ACCEPT screen statement to indicate the position of the visible cursor upon termination. (See Cursor locator.)

17. Data-name-3 shall be updated during the execution of an ACCEPT screen statement as described in CRT status.

SWITCH Clause

General format

Switch-name-1 [ON STATUS condition-name-1] [OFF STATUS condition-name-2]
Switch-name-1 [OFF STATUS condition-name-2] [ON STATUS condition-name-1]

Where switch-name-1 is:

SWITCH integer-1
| SWITCH nonnumeric-literal-1
| SYSTEM-SWITCH-1
| SYSTEM-SWITCH-2
| SYSTEM-SWITCH-3
| SYSTEM-SWITCH-4
| SYSTEM-SWITCH-5
| SYSTEM-SWITCH-6
| SYSTEM-SWITCH-7

| SYSTEM-SWITCH-8
| SYSTEM-SWITCH-9
| SYSTEM-SWITCH-10
| SYSTEM-SWITCH-11
| SYSTEM-SWITCH-12
| SYSTEM-SWITCH-13
| SYSTEM-SWITCH-14
| SYSTEM-SWITCH-15
| SYSTEM-SWITCH-16
| SYSTEM-SWITCH-17
| SYSTEM-SWITCH-18
| SYSTEM-SWITCH-19
| SYSTEM-SWITCH-20
| SYSTEM-SWITCH-21
| SYSTEM-SWITCH-22
| SYSTEM-SWITCH-23
| SYSTEM-SWITCH-24
| SYSTEM-SWITCH-25
| SYSTEM-SWITCH-26

Syntax rules

1. SWITCH integer-1 refers to the SWITCH numbered integer-1.
2. SWITCH nonnumeric-literal-1 refers to nonnumeric-literal-1 between "A" and "Z", or "a" and "z"; SWITCH "A" is SWITCH-1, …, SWITCH "Z" is SWITCH-26.
3. Condition-name-1 refers to the new condition name which will be true when the given switch is in the ON state.
4. Condition-name-2 refers to the new condition name which will be true when the given switch is in the OFF state.

General rules

1. The condition-name-1 and condition-name-2 defined to reference the given switch may be treated like any level 88 condition-name.
2. The SWITCH defaults to the OFF state.
3. To turn the SWITCH to the ON state, a setting must be made at runtime. Program parameters "s1" … "s26" may be set to "true" to set the switch to true. Command line parameters /A to /Z will set SYSTEM-SWITCH-1 to SYSTEM-SWITCH-26. Command line parameters /1 to /9 will set SYSTEM-SWITCH-1 to SYSTEM-SWITCH-9.
4. Certain aliases exist for the switches for compatibility with more platforms:

Name	Synonyms	
SYSTEM-SWITCH-1	SWITCH-1	UPSI-1
SYSTEM-SWITCH-2	SWITCH-2	UPSI-2
SYSTEM-SWITCH-3	SWITCH-3	UPSI-3
SYSTEM-SWITCH-4	SWITCH-4	UPSI-4
SYSTEM-SWITCH-5	SWITCH-5	UPSI-5
SYSTEM-SWITCH-6	SWITCH-6	UPSI-6

Name	Synonyms
SYSTEM-SWITCH-7	SWITCH-7 UPSI-7
SYSTEM-SWITCH-8	SWITCH-8 UPSI-8 SYSTEM-SHUTDOWN
SYSTEM-SWITCH-9	SWITCH-9 UPSI-9
SYSTEM-SWITCH-10	SWITCH-10 UPSI-0 SYSTEM-SWITCH-0 SWITCH-0
SYSTEM-SWITCH-11	SWITCH-11
SYSTEM-SWITCH-12	SWITCH-12
SYSTEM-SWITCH-13	SWITCH-13
SYSTEM-SWITCH-14	SWITCH-14
SYSTEM-SWITCH-15	SWITCH-15
SYSTEM-SWITCH-16	SWITCH-16
SYSTEM-SWITCH-17	SWITCH-17
SYSTEM-SWITCH-18	SWITCH-18
SYSTEM-SWITCH-19	SWITCH-19
SYSTEM-SWITCH-20	SWITCH-20
SYSTEM-SWITCH-21	SWITCH-21
SYSTEM-SWITCH-22	SWITCH-22
SYSTEM-SWITCH-23	SWITCH-23
SYSTEM-SWITCH-24	SWITCH-24
SYSTEM-SWITCH-25	SWITCH-25
SYSTEM-SWITCH-26	SWITCH-26

IMPLEMENTOR Clause

General format

predefined-mnemonic IS mnemonic-name-1
mnemonic-name-2 IS mnemonic-name-2
nonnumeric-literal-1 IS INITIAL CLASS
nonnumeric-literal-2 IS INITIAL LINKAGE
nonnumeric-literal-3 IS FILE CHARACTER
NUMERIC SIGN TRAILING [SEPARATE]
NUMERIC SIGN LEADING [SEPARATE]
CONSOLE IS CRT
CRT IS CONSOLE
ARGUMENT-VALUE IS COMMAND-LINE

Where predefined-mnemonic is:

CONSOLE
| PRINTER
| SYSOUT
| SYSIN
| SYSTEM
| SYSERR
| SERVLETOUT
| SERVLETIN
| ARGUMENT-NUMBER
| ARGUMENT-VALUE
| ENVIRONMENT-NAME
| ENVIRONMENT-VALUE

| PAGE
| COMMAND-LINE
| I-O-FEEDBACK
| DATA-AREA
| OPEN-FEEDBACK
| LOCAL-DATA
| PIP-DATA
| ATTRIBUTE-DATA

Syntax rules

1. Mnemonic-name-1 is a user variable.
2. The phrases 'CONSOLE IS CRT', 'CRT IS CONSOLE', 'ARGUMENT-VALUE IS COMMAND-LINE' are treated as commentary.
3. Nonnumeric-literal-1 must be a Java class name.
4. Nonnumeric-literal-2 must be a native dynamic link library or native shared object library.
5. Nonnumeric-literal-3 must be one character in length.
6. nonnumeric-literal-1 IS INITIAL CLASS
7. nonnumeric-literal-2 IS INITIAL LINKAGE
8. nonnumeric-literal-3 IS FILE CHARACTER

General rules

1. Default mnemonic-name-1 as a user defined mnemonic name for predefined-mnemonic or for another user defined mnemonic-name-2.
2. Certain synonyms for the predefined mnemonics exist:

Name	Synonym
PRINTER	PRINT, PRINTER-1
SYSOUT	SYSLST, SYSLIST, SYSOUT-FLUSH, SYSPCH, SYSPUNCH
SYSIN	SYSIPT
SYSTEM	REQUESTOR
SERVLETOUT	SERVLET-OUT, SESSION-OUT, SESSIONOUT
SERVLETIN	SERVLET-IN, SESSION-IN, SESSIONIN

3. The CONSOLE device is the graphical CONSOLE where possible. If no graphics are available, a CURSES (text terminal device) is used. If CURSES is not available, then it defaults to SYSOUT.
4. The PRINTER device maps to the Java printing capabilities for the platform. The PRINTER always refers to a graphical printer, usually the default printer for the platform.
5. SYSOUT refers to the standard output stream.
6. SYSIN refers to the standard input stream.
7. SYSTEM refers to SYSOUT for output, SYSIN for input.

8. SYSERR refers to the standard error output stream.
9. SERVLETOUT refers to the default session output stream. This is the output stream for any runtime environment, such as Servlets, TCP/IP session environment, etc. The SERVLETOUT device is different for each COBOL session.
10. SERVLETIN refers to the default session input stream. This is the input stream for any runtime environment, such as Servlets, TCP/IP session environment, etc. The SERVLETIN device is different for each COBOL session.
11. ARGUMENT-NUMBER is used to set the ARGUMENT-NUMBER for X/Open argument handling. Avoid in new code as this cannot be made thread-safe. See SET for better capabilities.
12. ARGUMENT-VALUE is used to set and get the ARGUMENT-VALUE for X/Open argument handling. Avoid in new code as this cannot be made thread-safe. See SET for better capabilities.
13. ENVIRONMENT-NAME is used to set the ENVIRONMENT-NAME for X/Open environment variable handling. Avoid in new code as this cannot be made thread-safe. See SET for better capabilities. This actually refers to program parameters rather than environment variables.
14. ENVIRONMENT-VALUE is used to set the ENVIRONMENT-VALUE for X/Open environment variable handling. Avoid in new code as this cannot be made thread-safe. See SET for better capabilities. This actually refers to program parameters rather than environment variables.
15. Nonnumeric-literal-1 is loaded when the COBOL program is loaded, as if by the Java sequence Class.forName(nonnumeric-literal-1).
16. Nonnumeric-literal-2 is loaded as a native library, as if by the Java sequence System.loadLibrary(nonnumeric-literal-2).
17. Nonnumeric-literal-3 is set as the default implicit file character, the character in a filename separating the directory paths and the final filename from one another. This character is replaced by the actual platform separator character in filenames dynamically. By default, this character is '\' in Windows, and '/' in Posix. When "\" IS FILE CHARACTER is specified, the name "mydir\myfile" will automatically be converted to "mydir/myfile" in Posix. All programs in a run unit must use the same value for the file character.
18. The NUMERIC clauses set the default location of the numeric sign. Normally, the default is TRAILING, but this clause may be used to override the normal default.

ALPHABET Clause

Define a user name for an alphabet as a sequence of characters or as a pre-defined alphabet.

ALPHABET alphabet-1 IS alphabet-phrase

Where alphabet-phrase is:

STANDARD-1
| STANDARD-2
| NATIVE
| EBCDIC
| character-specifications

Where character-specifications is:

{character-specification [ALSO character-specification]…}…

Where character-specification is:

{integer-1 [THROUGH|THRU integer-2]}
|{nonnumeric-literal-1 [THROUGH|THRU nonnumeric-literal-2]}
|{SPACE|SPACES}
|{QUOTE|QUOTES}

Syntax rules

1. Each character specification is assigned to the sequence in order.
2. Integer-1 and integer-2 shall be unsigned.
3. THROUGH and THRU are synonymous.
4. Nonnumeric-literal-1 must be one character in length if THROUGH is specified; it is treated the same as if its ordinal value were integer-1.
5. Nonnumeric-literal-2 must be one character in length if THROUGH is specified; it is treated the same as if its ordinal value were integer-2.
6. Each character-specification-1 ALSO character-specification-2 has both integer-1 and integer-2 at the same position in the collating sequence.
7. Whenever THROUGH is specified, all characters starting from integer-1 and ending at integer-2, inclusive, are added in order to the class.
8. SPACES and QUOTES are treated as single character items.

SYMBOLIC Clause

Define symbolic names for characters.

General format

SYMBOLIC CHARACTERS
[{symbolic-character-1}… [IS|ARE] {integer-1}…]

Syntax rules

1. A given symbolic-character-1 shall be specified only once within the SYMBOLIC CHARACTER clause of this SPECIAL-NAMES paragraph.
2. The relationship between each symbolic-character-1 and the corresponding integer-1 is by position in the SYMBOLIC CHARACTERS clause. The first symbolic-character-1 is paired with the first integer-1, the second symbolic-character-1 with the second integer-1, and so on.

3. There shall be a one-to-one correspondence between occurrences of symbolic-character-1 and occurrences of integer-1.

General rules

1. Symbolic-character-1 may be used afterwards in the program to refer to the character integer-1.

CLASS Clause

Define a class of characters, not to be confused with the object oriented class concept.

General format

CLASS class-name-1 character-specifications

Where character-specifications is:

{character-specification [ALSO character-specification]…}…

Where character-specification is:

{integer-1 [THROUGH|THRU integer-2]}
|{nonnumeric-literal-1 [THROUGH|THRU nonnumeric-literal-2]}
|{SPACE|SPACES}
|{QUOTE|QUOTES}

Syntax rules

1. Each character specification is assigned to the sequence in order.
2. Integer-1 and integer-2 shall be unsigned.
3. THROUGH and THRU are synonymous.
4. Nonnumeric-literal-1 must be one character in length if THROUGH is specified; it is treated the same as if its ordinal value were integer-1.
5. Nonnumeric-literal-2 must be one character in length if THROUGH is specified; it is treated the same as if its ordinal value were integer-2.
6. Each character-specification-1 ALSO character-specification-2 has both integer-1 and integer-2 at the same position in the collating sequence.
7. Whenever THROUGH is specified, all characters starting from integer-1 and ending at integer-2, inclusive, are added in order to the class.
8. SPACES and QUOTES are treated as single character items.

General rules

1. Class-name-1 becomes a class of characters, suitable for specification as a COLLATING SEQUENCE or in validity checking of COBOL data.

CURRENCY Clause

Replace the default currency character ($) with an alternative currency character, usable in the COBOL source code PICTURE strings.

General format

CURRENCY SIGN nonnumeric-literal-1

Syntax rules

Nonnumeric-literal-1 must be a nonnumeric-literal 1 character in length.

General rules

The first character of nonnumeric-literal-1 becomes the new currency symbol used in the COBOL source code from that point onwards.

DECIMAL-POINT Clause

Interchange the operation of the decimal point (.) and comma (,) characters for basic internationalization.

General format

DECIMAL-POINT IS COMMA

COMMA IS DECIMAL-POINT

DECIMAL-POINT IS nonnumeric-literal-1

COMMA IS nonnumeric-literal-2

Syntax rules

1. Nonnumeric-literal-1 must be a nonnumeric-literal 1 character in length.
2. Nonnumeric-literal-2 must be a nonnumeric-literal 1 character in length.

General rules

1. DECIMAL-POINT IS COMMA, or COMMA IS DECIMAL-POINT interchange the usage of the decimal-point (.) and comma (,) characters in the COBOL source code.
2. DECIMAL-POINT is nonnumeric-literal-1 assigns the first character of nonnumeric-literal-1 to be the replacement decimal-point (.) in the COBOL source code.
3. COMMA is nonnumeric-literal-2 assigns the first character of nonnumeric-literal-2 to be the replacement comma (,) in the COBOL source code.

CURSOR Clause

Set the screen section CURSOR identifier. This may also be defined in-line with the variable identifier-1.

General format

CURSOR identifier-1

Syntax rules

identifier-1 is used to hold the CURSOR information for the text and graphical screen section.

General rules

See Chapter 8, Input/Output, for a full description.

CRT-STATUS Clause

Set the screen section CRT STATUS identifier. This may also be defined in-line with the variable identifier-1.

General format

CRT STATUS identifier-1

Syntax rules

identifier-1 is used to hold the CRT STATUS for the text and graphical screen section.

General rules

See Chapter 8, Input/Output, for a full description.

SCREEN CONTROL Clause

Set the screen section SCREEN CONTROL identifier. This may also be defined in-line with the variable identifier-1.

General format

SCREEN CONTROL identifier-1

Syntax rules

identifier-1 is used to hold the SCREEN CONTROL information for the graphical screen section.

General rules

See Chapter 8, Input/Output, for a full description.

EVENT STATUS Clause

Set the screen section EVENT STATUS identifier. This may also be defined in-line with the variable identifier-1.

General format

EVENT STATUS identifier-1

Syntax rules

identifier-1 is used to hold the EVENT STATUS information for the graphical screen section.

General rules

See Chapter 8, Input/Output, for a full description.

CALL-CONVENTION Clause

The CALL-CONVENTION clause sets up a user name for a calling convention number. The calling convention number is passed through to native code, or used by a Java class accessible by Heirloom, to determine how the native routine should be called. The usage is system dependent, dependent upon the implementation of native code for the platform and how it ascribes meanings to these values, but the usage generally follows Micro Focus conventions.

General format

CALL-CONVENTION integer-1 IS call-convention-1

Syntax rules

1. Call-convention-1 is usable as a call-convention representing calling convention integer-1.

   The default meaning for the bits of integer-1 is described by the following table. Not all bits will be meaningful on all platforms. These values may be added for a cumulative effect.

Parameters Right to Left	0
Parameters Left to Right	1
Parameters Removed from Stack by Caller	0
Parameters Removed from Stack by Called	2
Return Code Updated on Exit	0
Return Code Not Updated on Exit	4
Normal Linking Behavior	0
Call Resolved at Link Time	8
OS/2 OptLink	16
Thunked to 16 Bit	32
NT STDCALL	64
Java Unsafe (direct pointers to COBOL memory)	128

2. Typical calling convention sets in Windows are:

C	0
PASCAL	3
WINAPI (Old)	11

OS2API	11
STDCALL	64
UNSAFE	128

3. The default calling convention is STDCALL. The default is affected by the DLL-CONVENTION program parameter; when 0, C is used; when 1, STDCALL is used; when 2, UNSAFE is used.

4. Java normally does not allow access to the heap storage used for object values, thus when calling a C library the runtime prepares parameters in advance of the invocation of the library function. Most COBOL numerics are converted to C data types int, long, float, double or char *. They are converted to big-endian or little-endian format regardless of the COBOL data types used to store them. A temporary is allocated in the C runtime heap (malloc()) and data copied from the Java (COBOL) variable space into it. It is these that are passed to the C library function. For BY REFERENCE parameters, the values are copied back to Java (COBOL) variable space upon return from the library function and the temporary storage is freed. This is the safest form of action when Java and non-Java programs are mixed. The UNSAFE calling convention changes this. The Java Unsafe API is intended to be used by JVM internal operations, performance enhancing and analysis tools. Unsafe allows access to the actual pointers of Java variables. When the UNSAFE calling convention or -DDLL-CONVENTION=2 is specified these pointers are passed to the C library functions. This gives the libraries direct access to the linear memory spaced used for COBOL variable storage. You must use -cache:disable (the default) so that COBOL operations reload the memory that may be modified by the C library and the called function must be aware of the big-endian/little-endian format necessary or specify COBOL datatype parameters (e.g., IBM (ASCII) “-dt 7”) compiler option necessary to synchronize with the C library variable storage convention.

RETURN-CODE Clause

The RETURN-CODE clause sets up a user name for the special register RETURN-CODE which is the default storage area for data to be returned to the calling program or environment.

General format

{RETURN-CODE|CONDITION-CODE|PROGRAM-STATUS return-code-identifier} |

{ return-code-identifier RETURN-CODE|CONDITION-CODE|PROGRAM-STATUS}

Syntax rules

return-code-identifier is treated as the special register RETURN-CODE.

DYNAMIC CONFIGURATION Clause

The DYNAMIC CONFIGURATION is used to set program parameters at runtime from code.

General format

DYNAMIC CONFIGURATION {nonnumeric-literal-1 IS nonnumeric-literal-2}…

Syntax rules

1. Nonnumeric-literal-1 is the parameter name.
2. Nonnumeric-literal-2 is the parameter value.

General rules

1. The program parameter nonnumeric-literal-1 is set to the value nonnumeric-literal-2 at runtime.
2. This program parameter affects only Elastic COBOL settings checked after the program is initialized, and is available through the FUNCTION PARAMETER.
3. This setting is available across threads and sessions.

DYNAMIC ENVIRONMENT Clause

The DYNAMIC ENVIRONMENT is used to set System Properties at runtime from code.

General format

DYNAMIC ENVIRONMENT {nonnumeric-literal-1 IS nonnumeric-literal-2}…

Syntax rules

1. Nonnumeric-literal-1 is the parameter name.
2. Nonnumeric-literal-2 is the parameter value.

General rules

1. The program parameter nonnumeric-literal-1 is set to the value nonnumeric-literal-2 at runtime.
2. This System Property setting affects only Elastic COBOL settings checked after the program is initialized, and any Java classes which check the System Property setting after the program is initialized, and is available through the FUNCTION PARAMETER. This setting is useful for communication of default System Property settings to Java classes which are integrated with a Elastic COBOL program.
3. This setting is available across threads and sessions.

LINKAGE Clause

The LINKAGE clause is for the AS/400 only and is treated as commentary with a warning.

General format

LINKAGE TYPE {PROGRAM|PROCEDURE|nonnumeric-literal-1} FOR nonnumeric-literal-2

[USING linkage-using-item…]

where linkage-using-item is:

ALL [DESCRIBED]

Integer-1 [DESCRIBED]

Integer-1 THROUGH|THRU integer-2 [DESCRIBED]

REPOSITORY Paragraph

The REPOSITORY paragraph allows specification of class names that may be used within the scope of this program.

General format

REPOSITORY.

[repository-item]…

where repository-item is:

{ CLASS class-name-1
| CLASS class-name-1 [IS|AS] nonnumeric-literal-1
| object-class-id IS CLASS nonnumeric-literal-1
| COMPONENT object-class-id IS nonnumeric-literal-1
| PROPERTY object-class-id IS nonnumeric-literal-1
| EVENT event-id CLASS object-class-name event-descriptor…
}

where event-descriptor is:

{ ADD nonnumeric-literal-2
| DELETE nonnumeric-literal-3
| USING nonnumeric-literal-4
| FOR nonnumeric-literal-6…
}

Syntax rules

1. Class-name-1 shall not be specified multiple times. Each use of class-name-1 defines the class reference; if nonnumeric-literal-1 is specified, the external name reference is to nonnumeric-literal-1. The external name reference is a Java class name.
2. COMPONENT defines a class-name, documenting it as a visual component.
3. PROPERTY defines a class-name, documenting it as a property of a visual component.
4. EVENT defines an event handling class called event-id, a reference to the Java AWT 1.1+ event model. The event-descriptor define the adapter class.
    a. class-name-2 or nonnumeric-literal-7 names the interface class, e.g., "java.awt.event.ActionListener"
    b. nonnumeric-literal-2 specifies the name of the add method, e.g., "addActionListener". By default, this name is formed from other elements.
    c. nonnumeric-literal-3 specifies the name of the remove method, e.g., "removeActionListener". By default, this name is formed from other elements.
    d. nonnumeric-literal-4 specifies the class of event object, e.g., "java.awt.event.ActionEvent".
    e. Each repetition of nonnumeric-literal-6 specifies an event method which compiles the interface named by object-class-name, which has a parameter of nonnumeric-literal-4. Each repetition of nonnumeric-literal-6 to complete the interface must be listed in order to correctly compile and link the Elastic COBOL executable at the Java stage. The nonnumeric-literal-6 is the event name for referencing in evaluation of events, e.g., "actionPerformed".

General rules

1. Class-name-1 is the name of a class that may be used throughout the scope of the program.

2. The literal in the AS phrase, if specified, is the name that is externalized to the operating environment and used to import from the operating environment. The literal phrase should always be used. The operating environment is the Java namespace.
3. Class-name-1 is used in the DATA DIVISION as the type for USAGE OBJECT REFERENCE, and may be used in INVOKE to construct objects and INVOKE static methods.

FIGURATIVE-CONSTANTS Paragraph

General format

FIGURATIVE-CONSTANTS.

{identifier-1 nonnumeric-literal-1} …

Syntax rules

1. identifier-1 shall be a new figurative constant name.
2. Nonnumeric-literal-1 shall be the contents of identifier-1.

General rules

The FIGURATIVE-CONSTANTS paragraph is present for Wang compatibility and should not be used in new code.

INPUT-OUTPUT Section

The INPUT-OUTPUT section deals with the information needed to control transmission and handling of data between external media and the object program.

General format

[INPUT-OUTPUT SECTION.]

[file-control-paragraph]

[i-o-control-paragraph]

Syntax rules

The input-output section may be specified in a program.

FILE-CONTROL Paragraph

The FILE-CONTROL paragraph specifies file-related information.

General format

FILE-CONTROL.

select-clause

[file-control-entry]…

where file-control-entry is:

Format 1 (indexed):

assign-clause
|access-clause
|reserve-clause
|record-key-clause
|alternate-record-key-clause
|file-status-clause
|organization-clause
|recording-mode-clause
|password-clause
|control-area-clause
|data-size-clause
|index-size-clause
|nodisplay-clause
|pfkeys-clause
|cursor-column-clause
|file-locking-clause
|compression-clause
|encryption-clause

Format 2 (relative):

assign-clause
|access-clause
|reserve-clause
|record-key-clause
|alternate-record-key-clause
|file-status-clause
|organization-clause
|recording-mode-clause
|password-clause
|control-area-clause
|data-size-clause
|index-size-clause
|nodisplay-clause
|pfkeys-clause
|cursor-column-clause
|file-locking-clause
|compression-clause
|encryption-clause

Format 3 (sequential):

assign-clause
|access-clause
|padding-clause
|reserve-clause
|record-key-clause
|alternate-record-key-clause

|file-status-clause
|organization-clause
|record-delimiter-clause
|recording-mode-clause
|password-clause
|control-area-clause
|data-size-clause
|index-size-clause
|nodisplay-clause
|pfkeys-clause
|cursor-column-clause
|file-locking-clause
|compression-clause
|encryption-clause

Format 4 (sort-merge):
assign-clause
|organization-clause
.

Syntax rules ALL FORMATS

1. The SELECT clause shall be specified first in the file control entry. The clauses that follow the SELECT clause may appear in any order.
2. Each file-name in the data division shall be specified only once in the FILE-CONTROL paragraph. Each file-name specified in the SELECT clause shall have a file description entry in the data division of the same factory, function, method, object, or program.
3. Literal-1 shall be an alphanumeric literal and shall not be a figurative constant.
4. The meaning and rules for the allowable specification of device-name-1 and the value of literal-1 are defined by the implementor.
5. Data-name-1 shall reference an alphanumeric data item and shall not be subordinate to the file description entry for file-name-1.
6. Data-name-1 may be qualified.
7. Device-name-1 may be PRINTER, DISPLAY, or KEYBOARD.

FORMATS 1 AND 2

The MULTIPLE phrase shall be omitted for a file connector for which ACCESS MODE IS SEQUENTIAL is specified or implied.

Format 1 shall be specified only for an indexed file connector. The associated file description entry shall not be a sort-merge file description entry.

FORMAT 2

Format 2 shall be specified only for an relative file connector. The associated file description entry shall not be a sort-merge file description entry.

The RELATIVE clause shall be specified if the DYNAMIC or RANDOM phrase of the ACCESS clause is specified.

FORMAT 3

Format 3 shall be specified only for a sequential file connector. The associated file description entry shall not be a sort-merge file description entry.

FORMAT 4

Format 4 shall be specified only for a sort-merge file connector. The associated file description entry shall be a sort-merge file description entry.

General rules

ALL FORMATS

Format 4 shall be specified only for a sort-merge file connector. The associated file description entry shall be a sort-merge file description entry.

1. If the file connector referenced by file-name-1 is an external file connector (see EXTERNAL clause),all file control entries in the run unit that reference this file connector shall have:
    a. The same specification for the OPTIONAL phrase.
    b. A consistent specification for data-name-1, device-name-1, and literal-1 in the ASSIGN clause. The operating system determines the consistency rules for data-name-1, device-name-1, and literal-1.
    c. The same value for integer-1 in the RESERVE clause.
    d. The same organization.
    e. The same access mode.
    f. The same specification of alphabets for the COLLATING SEQUENCE clause.
    g. The same value for the PADDING CHARACTER clause.
    h. The same external data item for data-name-7 in the RELATIVE KEY clause.
    i. The same data description entry for data-name-5 and each data-name-6 as well as their relative location within the associated record.
    j. The same data description entry for data-name-2 and each data-name-3 as well as their relative location within the associated record, the same number of alternate record keys, and the same DUPLICATES phrase.
    k. The same sharing mode.
    l. The same lock mode and the same choice of either single record locking or multiple record locking.
2. The OPTIONAL phrase applies only to files opened in the input, I-O, or extend mode. Its specification is required for files that are not necessarily present each time the object program is executed.
   Elastic COBOL follows ANSI standard in requiring the OPTIONAL keyword for optional files. To match certain other COBOL implementations, Elastic COBOL supports the $SET OPTIONAL-FILE directive to default to OPTIONAL. NOT OPTIONAL may be used to override to the original behavior in this case.
3. The ASSIGN clause specifies the association of the file referenced by file-name-1 to a storage medium referenced by device-name-1, literal-1, or the content of

the data item referenced by data-name-1. The association occurs at the time of execution of an OPEN, SORT, or MERGE statement that referenced file-name-1, according to the following rules:

a. The value literal-1, or the data–name-1's contents, are used as the external file reference name.

b. This external file name may be a virtual device name, in which case the external file name begins with a protocol-name:. The protocol-name specifies the name of the virtual device driver provided by Elastic COBOL, such as socket or printer.

c. Protocol-names are checked only at runtime, not at compile-time, as the protocols present for various virtual devices vary between the various Elastic COBOL runtimes. An Applet-specific runtime, for instance, may not possess the server protocol, because there cannot be an applet server.

d. One-letter protocols are assumed to be drive letters and passed through without redirection (e.g., C: or D:).

If the association cannot be made because the content of the data item referenced by data-name-1 is not consistent with the specification for device-name-1 or literal-1, the OPEN, SORT, or MERGE statement is unsuccessful.

FORMAT 1

The indexed format defines a file connector for an indexed file.

FORMAT 2

The relative format defines a file connector for a relative file.

FORMAT 3

The sequential format defines a file connector for a sequential file.

FORMAT 4

The sort-merge format defines a file connector for a sort-merge file.

SELECT Clause

File Types: INDEXED, RELATIVE, SEQUENTIAL, SORT-MERGE

General format

SELECT [[NOT] OPTIONAL] file-name-1

Syntax rules

File-name-1 must be a valid identifier name in COBOL. This is the file-name referred to in all other documentation.

General rules

1. The SELECT clause assigns the internal, program source code name file-name-1 to the external file reference.
2. The SELECT clause is required as the first clause in all file types.

3. If the file may not be present at time of OPEN, and the OPEN will not be OUTPUT, then OPTIONAL should be specified; the OPTIONAL will allow the OPEN to succeed for a missing file, but cause the first READ to fail.
4. NOT OPTIONAL is treated as commentary.

ASSIGN Clause

File Types: INDEXED, RELATIVE, SEQUENTIAL

General format

ASSIGN [TO|USING] assign-name

Where assign-name is:

[VARYING assign-name
| DISK assign-name
| INPUT assign-name
| OUTPUT assign-name
| INPUT-OUTPUT assign-name
| SORT assign-name
| SORTMERGE assign-name
| MERGE assign-name
| RANDOM assign-name
| CONSOLE assign-name
]
| Nonnumeric-literal-1
| DYNAMIC
| DISK
| PRINTER
| DISPLAY
| KEYBOARD
| INPUT
| OUTPUT
| INPUT-OUTPUT
| SORT
| SORT-MERGE
| EXTERNAL word-1
| ENVIRONMENT word-1
| Nonnumeric-literal-2 nonnumeric-literal-3
| PRINTER nonnumeric-literal-4
| DISPLAY nonnumeric-literal-5
| KEYBOARD nonnumeric-literal-5

Syntax rules

DYNAMIC specifies that a file dialog will prompt the user for the external filename.

DISK|INPUT|OUTPUT|INPUT-OUTPUT|SORT|SORT-MERGE specifies that the external filename will be constructed programmatically from the internal filename, or specified later by a FILE-ID clause.

PRINTER resolves to "line:printer:graphics".

DISPLAY resolves to "console:"

KEYBOARD resolves to "console:"

EXTERNAL|ENVIRONMENT word-1 resolves to "env:word-1".

Nonnumeric-literal-1 is the external filename; this external filename will first be resolved through the Elastic COBOL protocols at runtime before the final external filename is known, allowing virtual devices and file system information to be granted to the runtime.

If nonnumeric-literal-3 is "DISK", then nonnumeric-literal-2 is the external filename. If nonnumeric-literal-3 is "DISPLAY", then "console:nonnumeric-literal-2" is used as the Wang style external filename.

PRINTER nonnumeric-literal-4 resolves to "line:printer:nonnumeric-literal-4"

DISPLAY|KEYBOARD nonnumeric-literal-5 resolves to "console:nonnumeric-literal-5".

ACCESS MODE Clause

The ACCESS MODE clause specifies the order in which records are to be accessed in the file.

General format

ACCESS MODE IS {DYNAMIC|RANDOM|SEQUENTIAL]

Syntax rules

1. The ACCESS MODE IS RANDOM clause shall not be specified for file-names specified in the USING or GIVING phrase of a SORT or MERGE statement or for a sequential file.
2. The DYNAMIC and RANDOM phrases shall not be specified for a sequential file.

General rules

1. If the ACCESS MODE clause is not specified, sequential access is assumed.
2. If the access mode is sequential, records in the file are accessed in the sequence dictated by the file organization:
    a. For sequential files this sequence is specified by predecessor-successor record relationships established by the execution of WRITE statements when the file is created or extended.
    b. For relative files this sequence is the order of ascending relative record numbers of existing records in the file.
    c. For indexed files this sequence is ascending within a given key of reference according to the collating sequence of the file.

3. If the access mode is random:
    a. For a relative file, the value of a relative key data item indicates the record to be accessed.
    b. For an indexed file, the value of a record key data item indicates the record to be accessed.
4. If the access mode is dynamic, records in the file may be accessed sequentially and/or randomly.
5. If the associated file connector is an external file connector, every file control entry in the run unit that is associated with that file connector shall specify the same access mode.

ALTERNATE RECORD KEY Clause

The ALTERNATE RECORD KEY clause specifies an alternate record key access path to the records in an indexed file.

File Types: INDEXED

General format

ALTERNATE {RECORD KEY|RECORD-KEY} data-name-5|split-name-clause
[WITH {DUPLICATES IN ORDER}|{NO DUPLICATES}]

where split-name-clause is:

data-name-1 = data-name-2…
EXTERNAL = data-name-2…

Syntax rules

1. Data-name-1 may be qualified.
2. Data-name-1 shall be defined as a data item of category alphanumeric or national within a record description entry associated with the file-name to which the ALTERNATE RECORD KEY clause is subordinate. All occurrences of data-name-2 shall be of the same category.
3. Data-name-1 shall not reference a group item that contains a variable-occurrence data item.
4. Data-name-1 shall not reference an item whose leftmost character position corresponds to the leftmost character position of the prime record key, or of another alternate record key. This restriction does not apply in the case where either key is specified using the SOURCE phrase.
5. If the indexed file contains variable-length records, each data-name-1 and data-name-2 shall be contained within the first x character positions of the record, where x equals the minimum record size specified for the file. (See RECORD clause.)
6. If the file has more than one record description entry, data-name-1 and data-name-2 need only be described in one of these record description entries.

7. When defined as a split-name, data-name-1 is a name usable only as a keyname for START; this is the combination key of all of its data-name-2 components.

General rules

1. An ALTERNATE RECORD KEY clause specifies an alternate record key for the file with which this clause is associated.
2. Record-key-name-1 defines a record key consisting of the concatenation of all occurrences of data-name-2 in the order specified.
3. The data description of data-name-1 or data-name-2 as well as their relative location within a record shall be the same as that used when the file was created. The number of alternate record keys for the file shall also be the same as that used when the file was created.
4. The DUPLICATES phrase specifies that the value of the associated alternate record key may be duplicated within any of the records in the file. If the DUPLICATES phrase is not specified, the value of the associated alternate record key shall not be duplicated among any of the records in the file.
5. If the file has more than one record description entry, data-name-1 or data-name-2 need only be described in one of these record description entries. The identical character positions referenced by data-name-1 or data-name-2 in any one record description entry are implicitly referenced in keys for all other record description entries of that file.
6. If the associated file connector is an external file connector, every file control entry in the run unit that is associated with that file connector shall specify the same data description entry for data-name-1 or data-name-2, the same relative location within the associated record, the same number of alternate record keys, and the same DUPLICATES phrase.

COMPRESSION Clause

Enable compression for the file.

General format

WITH COMPRESSION [CONTROL VALUE integer-1]

Syntax Notes

1. The COMPRESSION request flag is passed to the runtime, but currently not supported by any file system.
2. The integer-1 value is a percentage from 0 (none) to 100 (maximum), with the value 1 being a special meaning default level of compression.

ENCRYPTION Clause

Enable encryption for the file.

General format

WITH ENCRYPTION

Syntax Notes

The ENCRYPTION request flag is passed to the runtime, but currently not supported by any file system.

COMPRESSION Clause

Enable compression for the file.

General format

WITH COMPRESSION [CONTROL VALUE integer-1]

Syntax Notes

1. The COMPRESSION request flag is passed to the runtime, but currently not supported by any file system.
2. The integer-1 value is a percentage from 0 (none) to 100 (maximum), with the value 1 being a special meaning default level of compression.

CONTROL AREA Clause

CONTROL-AREA is an AS/400 COBOL clause, currently not supported.

General format

{CONTROL AREA | CONTROL-AREA} data-name-13

CURSOR COLUMN Clause

CURSOR COLUMN is Wang specific and treated as commentary.

File Types: SEQUENTIAL

General format

CURSOR COLUMN data-name-15

DATA SIZE Clause

The DATA SIZE clause is vendor specific clauses and treated as commentary.

General format

DATA SIZE integer

ENCRYPTION Clause

Enable encryption for the file.

General format

WITH ENCRYPTION

Syntax Notes

1. The ENCRYPTION request flag is passed to the runtime, but currently not supported by any file system.

INDEX SIZE Clause

The INDEX SIZE clause is vendor specific clauses and treated as commentary.

General format

INDEX SIZE integer

NODISPLAY Clause

The NODISPLAY clause is vendor specific clauses and treated as commentary.

General format

NODISPLAY

PFKEYS Clause

PFKEYS is Wang specific and treated as commentary.

File Types: SEQUENTIAL

General format

PFKEYS data-name-14

FILE LOCKING Clause

Enable file and record locking for the file. This specifies the level of sharing allowed for the file and its record contents. This definition of file locking and sharing is used in conjunction with the file locking and sharing specified by the OPEN statement to determine the final locking and sharing options for the file.

File Types: INDEXED, RELATIVE, SEQUENTIAL

General format

[[LOCK MODE IS {MANUAL|AUTOMATIC}

[WITH LOCK ON [MULTIPLE] {RECORD|RECORDS}]] |

LOCK MODE EXCLUSIVE [WITH MASS-UPDATE]]]

Mode	Syntax
X:	LOCK MODE EXCLUSIVE
XU:	LOCK MODE EXCLUSIVE WITH MASS-UPDATE
LM:	LOCK MODE MANUAL WITH LOCK ON MULTIPLE [RECORD|RECORDS]
L:	LOCK MODE MANUAL WITH LOCK ON {RECORD|RECORDS}
A:	LOCK MODE AUTO WITH LOCK ON RECORD

Mode	Syntax
AM:	LOCK MODE AUTO WITH LOCK ON MULTIPLE [RECORD|RECORDS]
A:	LOCK MODE AUTO
L:	LOCK MODE MANUAL
S:	SHARING WITH ALL OTHER
N:	SHARING WITH NO OTHER
R:	SHARING WITH READ ONLY
LMB:	LOCK MODE MANUAL WITH LOCK ON MULTIPLE ROLLBACK
LB:	LOCK MODE MANUAL WITH LOCK ON ROLLBACK
AB:	LOCK MODE AUTO WITH LOCK ROLLBACK

Key

Mode	Name
X	EXCLUSIVE
U	MASS-UPDATE
M	MULTIPLE
L	MANUAL
A	AUTOMATIC
B	ROLLBACK
S	SHARING ALL OTHERS
N	SHARING NO OTHERS
R	SHARING READ ONLY

General rules

1. If the LOCK MODE clause is omitted from a file control entry, record locks have no effect for the associated file connector.
2. If a file is open in the sharing with no other mode, the LOCK MODE clause has no effect. Otherwise, the LOCK MODE clause has the effects described in the general rules that follow.
3. If the AUTOMATIC phrase is specified, the lock mode is automatic. Records are locked:
   a. when any READ statement for which neither the IGNORING phrase nor the NO LOCK phrase is specified is executed, and
   b. when a REWRITE or WRITE statement for which the LOCK phrase is specified is executed.
4. If the MANUAL phrase is specified, the lock mode is manual. Records locks are obtained only when the LOCK phrase is explicitly specified on an I-O statement.
5. Single record locking is specified explicitly by the LOCK ON phrase without the MULTIPLE phrase and implicitly when the LOCK MODE clause is specified with the LOCK ON phrase omitted. Single record locking allows only one record of a file to be locked at a given time through a single file connector. Completion of the successful execution of a statement that locks a record releases any previously locked record in that file for that file connector.
6. If the MULTIPLE phrase is specified in the LOCK ON phrase, then multiple record locking is said to have been specified and a file connector is permitted to have more than one record of a file locked. A file connector that has specified multiple record locking for a file may hold a number of record locks for that file simultaneously. This prevents other file connectors from accessing any member of the set of locked records, but will not deny them access to records that are not locked. The implementor shall specify the maximum number of record locks that may be held by a file connector and the maximum number of record locks that

may be held by a run unit. Both of these maximum numbers shall be at least one. Any I-O statement that attempts to obtain a record lock that would exceed either limit is unsuccessful and receives an I-O status that indicates that condition.

7. The setting of a record lock is part of the atomic operation of an I-O statement.
8. If MASS-UPDATE is specified, write caching is enabled.
9. EXCLUSIVE implies SHARING READ ONLY for INPUT files, SHARING NO OTHERS for other files.

FILE STATUS Clause

The FILE STATUS clause specifies a data item that contains the status of an input-output operation.

File Types: INDEXED, RELATIVE, SEQUENTIAL

General format

FILE STATUS data-name-4

Syntax rules

1. Data-name-1 may be qualified.
2. Data-name-1 shall be a two-character data item of the category alphanumeric, defined in the working-storage or linkage section.

General rules

If the FILE STATUS clause is specified, the data item referenced by data-name-1 is updated to contain the value of the I-O status for the file connector referenced by the subject of the entry when the I-O status associated with that file connector is updated as a result of an input-output statement.

NOTE - In the case where a file-name is global and data-name-1 is not, data-name-1 is updated by references to file-name in contained programs even though data-name-1 is a local name.

ORGANIZATION Clause

The ORGANIZATION clause specifies the logical structure of a file.

File Types: INDEXED, RELATIVE, SEQUENTIAL, SORT-MERGE

General format

ORGANIZATION {[BINARY|RECORD] SEQUENTIAL | RELATIVE | INDEXED | LINE SEQUENTIAL | TRANSACTION}

General rules

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

2. The SEQUENTIAL phrase specifies that the file organization is sequential. Sequential organization is a permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file. LINE SEQUENTIAL is a variant of SEQUENTIAL wherein records are stored as lines of text; do not include non-DISPLAY data in LINE SEQUENTIAL files.
3. The RELATIVE phrase specifies that the file organization is relative. Relative organization is a permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, that specifies the records logical ordinal position in the file.
4. The INDEXED phrase specifies that the file organization is indexed. Indexed organization is a permanent logical file structure in which each record is identified by the value of one or more keys within that record. When the ORGANIZATION clause is not specified, sequential organization is implied.
5. TRANSACTION is an AS/400 COBOL specific organization, currently not supported.

PADDING CHARACTER Clause

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

File Types: SEQUENTIAL

General format

PADDING CHARACTER data-name-1|literal-1

Syntax rules

1. Literal-1 shall be a one-character alphanumeric literal.
2. Data-name-1 may be qualified.
3. Data-name-1 shall be a one-character data item of category alphanumeric, defined in the working-storage or linkage section.

General rules

1. The PADDING CHARACTER clause specifies the character that is to be used for block padding on sequential files. During input operations, any portion of a block that exists beyond the last logical record and consists entirely of padding characters shall be bypassed. During input operations, a logical record that consists solely of padding characters shall be ignored. During output operations, any portion of a block that exists beyond the last logical record shall be filled entirely with padding characters.
2. If the PADDING CHARACTER clause is not applicable to the device type to which the file is assigned, the creation or recognition of padding characters shall not occur.
3. Literal-1 or the value of the data item referenced by data-name-1, at the time the OPEN statement that creates the file is executed, is used as the value of the padding character.

4. If the CODE-SET clause is specified for the file, conversion of the padding character specified by literal-1 or the content of data-name-1 is established for the file when the file is opened.
5. If the PADDING CHARACTER clause is not specified, the value used for the padding character shall be defined by the implementor.
6. If the associated file connector is an external file connector, all PADDING CHARACTER clauses in the run unit that are associated with that file connector shall have the same specifications. If data-name-1 is specified, it shall reference an external data item.
7. Padding character is treated as commentary.

PASSWORD Clause

PASSWORD is a vendor specific clause, treated as commentary.

General format

PASSWORD data-name-12

RECORD DELIMITER Clause

The RECORD DELIMITER clause indicates the method of determining the length of a variable-length record on the external medium.

File Types: SEQUENTIAL

General format

RECORD DELIMITER IS STANDARD-1

Syntax rules

The RECORD DELIMITER clause may be specified only for variable-length records.

NOTE - There are three ways variable-length records may be specified:

1. The RECORD clause is not specified and the implementor has specified that variable-length records are obtained in this circumstance.
2. The RECORD IS VARYING clause is specified.
3. The format 3 RECORD CONTAINS clause is specified with different from and to values for Elastic COBOL file format.
4. RECORDING MODE V is specified.

General rules

1. The RECORD DELIMITER clause indicates the method of determining the length of a variable-length record on the external medium. Any method used shall not be reflected in the record area or the record size used within the program.
2. If STANDARD-1 is specified, the external medium shall be a magnetic tape file.

3. If STANDARD-1 is specified, the method used for determining the length of a variable-length record is that specified in ISO 1001.
4. If feature-name-1 is specified, the method used for determining the length of a variable-length record is that associated with feature-name-1 by the implementor.
5. If the RECORD DELIMITER clause is not specified, the method used for determining the length of a variable-length record is specified by the implementor.
6. At the time of a successful execution of an OPEN statement, the record delimiter is the one specified in the RECORD DELIMITER clause in the file control entry associated with the file-name specified in the OPEN statement.
7. If the associated file connector is an external file connector, all RECORD DELIMITER clauses in the run unit that are associated with that file connector shall have the same specifications.
8. Record delimiter is treated as commentary.

RECORDING MODE Clause

The RECORDING MODE clause sets the preferred storage to be fixed or variable.

File Types: INDEXED, RELATIVE

General format

RECORDING MODE F|FIXED|V|VARIABLE

Syntax rules

1. FIXED and F are treated the same.
2. VARIABLE and V are treated the same.

General rules

1. FIXED forces the runtime to store records in fixed format.
2. VARIABLE forces the runtime to store records in variable format.

RECORD KEY Clause

The RECORD KEY clause specifies the prime record key access path to the records in an indexed file.

File Types: INDEXED

General format

{RECORD KEY|RECORD-KEY} data-name-1|split-name-clause

where split-name-clause is:

data-name-1 = data-name-2…

EXTERNAL = data-name-2…

Syntax rules

1. Data-name-1 may be qualified.
2. Data-name-1 shall reference a data item of category alphanumeric or category national within a record description entry associated with the file-name specified in this file control entry. All occurrences of data-name-2 shall be of the same category.
3. Data-name-1 shall not reference a group item that contains a variable-occurrence data item.
4. If the indexed file contains variable-length records, data-name-1 shall be contained within the first n character positions of the record, where n equals the minimum record size specified for the file. (See RECORD clause.)
5. Split-name-1 is a name usable only as a keyname for START; this is the combination key of all of its data-name-2 components.

General rules

1. The RECORD KEY clause specifies the prime record key for the file with which this clause is associated. The values of the prime record key shall be unique among records of the file.
2. If split-name is used, data-name-1 is a new identifier consisting of the concatenation of all occurrences of data-name-2 in the order specified.
3. The data description of data-name-1 or data-name-2 as well as their relative location within a record shall be the same as that used when the file was created.
4. If the file has more than one record description entry, data-name-1 or data-name-2 need only be described in one of these record description entries. The identical character positions referenced by data-name-1 or data-name-2 in any one record description entry are implicitly referenced as keys for all other record description entries of that file.
5. 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 shall specify the same data description entry for data-name-1 or data-name-2 with the same relative location within the associated record.

RELATIVE KEY Clause

The RELATIVE KEY clause identifies the data item that will contain the relative record number for accessing a relative file.

General format

{RELATIVE|ACTUAL} KEY IS data-name-1

Syntax rules

1. Data-name-1 may be qualified.
2. Data-name-1 shall reference an unsigned integer data item whose description does not contain the PICTURE symbol 'P'.

3. Data-name-1 shall not be defined in a record description entry subordinate to the associated file-name.

General rules

1. All records stored in a relative file are uniquely identified by relative record numbers. The relative record number of a given record specifies the record's logical ordinal position in the file. The first logical record has a relative record number of 1, and subsequent logical records have relative record numbers of 2, 3, 4, ...
2. The relative key data item associated with the execution of an input-output statement is the data item referenced by data-name-1; data-name-1 is used to communicate a relative record number between the user and the mass storage control system (MSCS).
3. If the associated file connector is an external file connector, every file control entry in the run unit that is associated with that file connector shall contain a RELATIVE KEY clause, data-name-1 shall reference an external data item, and the RELATIVE KEY clause in each associated file control entry shall reference that same external data item in each case.

RESERVE Clause

The RESERVE clause allows the user to specify the number of input-output areas allocated.

File Types: INDEXED, RELATIVE, SEQUENTIAL

General format

RESERVE {integer-1|NO} ALTERNATE {AREA|AREAS}

General rules

If the RESERVE clause is specified, the number of input-output areas allocated is equal to the value of integer-1.

RESERVE clause is treated as commentary.

SHARING Clause

The SHARING clause indicates that a file is to participate in file sharing and record locking. It specifies the degree of file sharing (or non-sharing) to be permitted for a file and whether record locks have an effect.

General format

SHARING WITH { ALL OTHER | NO OTHER | READ ONLY}

Mode	Syntax
X:	LOCK MODE EXCLUSIVE
XU:	LOCK MODE EXCLUSIVE WITH MASS-UPDATE
LM:	LOCK MODE MANUAL WITH LOCK ON MULTIPLE [RECORD|RECORDS]
L:	LOCK MODE MANUAL WITH LOCK ON {RECORD|RECORDS}
A:	LOCK MODE AUTO WITH LOCK ON RECORD

Mode	Syntax
AM:	LOCK MODE AUTO WITH LOCK ON MULTIPLE [RECORD|RECORDS]
A:	LOCK MODE AUTO
L:	LOCK MODE MANUAL
S:	SHARING WITH ALL OTHER
N:	SHARING WITH NO OTHER
R:	SHARING WITH READ ONLY
LMB:	LOCK MODE MANUAL WITH LOCK ON MULTIPLE ROLLBACK
LB:	LOCK MODE MANUAL WITH LOCK ON ROLLBACK
AB:	LOCK MODE AUTO WITH LOCK ROLLBACK

Key

Mode	Name
X	EXCLUSIVE
U	MASS-UPDATE
M	MULTIPLE
L	MANUAL
A	AUTOMATIC
B	ROLLBACK
S	SHARING ALL OTHERS
N	SHARING NO OTHERS
R	SHARING READ ONLY

General rules

The SHARING clause specifies the sharing mode to be used for the file unless it is overridden by the SHARING phrase of the OPEN statement. This clause also specifies whether record locks have an effect. Additional details are specified in Sharing mode.

I-O-CONTROL Paragraph

The I-O-CONTROL paragraph specifies that memory areas associated with different files are to be shared during file processing, record processing, or sort-merge processing.

General format

```
I-O-CONTROL.
[
        [same_clause]…
        [rerun_clause]…
        [multiple_file_clause]…
        [apply_clause]…
    [commitment_clause]…
.]
```

SAME Clause

The SAME clause specifies files for which memory areas are to be shared during file processing, record processing, or sort-merge processing.

General format

Format 1 (file-area):

[SAME AREA FOR file-name-1 {file-name-2}…] …

Format 2 (record-area):

[SAME RECORD AREA FOR file-name-1 {file-name-2}…] …

Format 3 (sort-merge-area):

[SAME {SORT|SORT-MERGE} AREA FOR file-name-1 {file-name-2} …] …

Syntax rules

1. SORT and SORT-MERGE are equivalent.
2. File-name-1 and file-name-2 shall be specified in the FILE-CONTROL paragraph of the program that contains this SAME clause.
3. File-name-1 and file-name-2 shall not reference an external file connector.
4. The files specified in a given SAME clause need not all have the same organization or access.
5. A given file-name that represents a report file may be specified in one file-area format SAME clause and shall not be specified in a record-area format or sort-merge-area format SAME clause.
6. A given file-name that represents a sort or merge file may be specified in one record-area format SAME clause and in one sort-merge-area SAME clause, and shall not be specified in a file-area format SAME clause.
7. A given file-name that represents a file other than a report file or a sort or merge file may be specified in one file-area format, in one record-area format, and in one or more sort-merge-area format SAME clauses.
8. At least one file-name specified in a sort-merge-area format SAME clause shall represent a sort or merge file.
9. If one or more file-names specified in a file-area format SAME clause are also specified in a record-area format SAME clause, all of the file-names specified in the file-area format SAME clause shall also be specified in the record-area format SAME clause. Additional file-names not specified in the file-area format SAME clause may be specified in the record-area format SAME clause.
10. If a file-name that represents a file other than a sort or merge file is specified in a file-area format SAME clause and in one or more sort-merge-area format SAME clauses, all of the file-names specified in that file-area format SAME clause shall also be specified in those sort-merge-area format SAME clause(s).

General rules

1. A file-area format SAME clause specifies that two or more files referenced by file-name-1, file-name-2 are to use the same memory area during processing. The area being shared includes all storage areas assigned to the files referenced by file-name-1, file-name-2. No more than one of these files may be in the open mode at a given time.
2. A record-area format SAME clause specifies that two or more files referenced by file-name, file-name-2 are to share a memory area for processing the current logical record. All of these files may be in the open mode at the same time,

except that only one file that is also specified in a file-area format SAME clause may be open at that time. A logical record in the shared memory area is a logical record of each file open in the output mode and of the most recently-read file open in the input mode. This is equivalent to an implicit redefinition of the area with records aligned on the leftmost character position.

3. A sort-merge-area format SAME clause specifies that memory is shared as follows:
   a. Any storage area allocated for the sorting or merging of a sort or merge file specified in a sort-merge-area format SAME clause is available for reuse in sorting or merging any of the other sort or merge files specified in that sort-merge-area format SAME clause.
   b. Storage areas assigned to files specified in a sort-merge-area format SAME clause that do not represent sort or merge files may be allocated as needed for sorting or merging the sort or merge files named in that sort-merge-area format SAME clause.
   c. Storage areas assigned to files specified in a sort-merge-area format SAME clause other than sort or merge files do not share the same storage area with each other.
4. During the processing of a SORT or MERGE statement that refers to a sort or merge file named in a sort-merge-area format SAME clause, any non-sort and non-merge files associated with file-names specified in that clause shall not be in the open mode.

RERUN Clause

The SAME clause specifies files for which memory areas are to be shared during file processing, record processing, or sort-merge processing.

General format

RERUN [ON {file-name-1|implementor-name-1}] EVERY
{integer-2 CLOCK-UNITS
| condition-name-1
| { {integer-1 RECORDS} | {[END OF] {REEL|UNIT}}} OF file-name-2}

Syntax rules

1. File-name-1 must be a sequentially organized file.
2. The END OF REEL/UNIT phrase may only be used if file-name-2 is a sequentially organized file. The definition of UNIT is determined by each implementor.
3. When either the integer-1 RECORDS phrase or the integer-2 CLOCK-UNITS phrase is specified, implementor-name-1 must be given in the RERUN clause.
4. More than one RERUN clause may be specified for a given file-name-2 subject to the following restrictions:
   a. When multiple integer-1 RECORDS phrases are specified, no two of them may specify the same file-name-2.

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

5. Only one RERUN clause containing the CLOCK-UNITS phrase may be specified.

General rules

1. The RERUN clause specifies when and where the rerun information is recorded. Rerun information is recorded in the following ways:

   a. If file-name-1 is specified, the rerun information is written on each reel or unit of an output file and the implementor specifies where, on the reel or file, the rerun information is to be recorded.

   b. If implementor-name is specified, the rerun information is written as a separate file on a device specified by the implementor.

2. There are seven forms of the RERUN clause, based on the several conditions under which rerun points can be established. The implementor must provide at least one of the specified forms of the RERUN clause.

   a. When either the END OF REEL or END OF UNIT phrase is used without the ON phrase. In this case, the rerun information is written on file-name-2 which must be an output file.

   b. When either the END OF REEL or END OF UNIT phrase is used and file-name-1 is specified in the ON phrase. In this case, the rerun information is written on file-name-1, which must be an output file. In addition, normal reel, or unit, closing functions for file-name-2 are performed. File-name-2 may either be an input or an output file.

   c. When either the END OF REEL or END OF UNIT phrase is used and implementor-name is specified in the ON phrase. In this case, the rerun information is written on a separate rerun unit defined by the implementor. File-name-2 may be either an input or output file.

   d. When the integer-1 RECORDS phrase is used. In this case, the rerun information is written on the device specified by implementor-name-1, which must be specified in the ON phrase, whenever an interval of time, calculated by an internal clock, has elapsed.

   e. When the condition-name-1 phrase is used and implementor-name-1 is specified in the ON phrase. In this case, the rerun information is written on the device specified by implementor-name-1 whenever a switch assumes as particular status as specified by condition-name-1. In this case, the associated switch must be defined in the SPECIAL-NAMES paragraph of the Configuration Section of the Environment Division. The implementor specified when the switch status is interrogated.

   f. When the condition-name-1 phrase is used and file-name-1 is specified in the ON phrase. In this case, the rerun information is written on file-name-1, which must be an output file, whenever a switch assumed a particular status as specified by condition-name-1. In this case, as in paragraph f above, the associated switch must be defined in the SPECIAL-NAMES paragraph of the Configuration Section of the Environment Division. The implementor specifies when the switch status is interrogated.

3. Elastic COBOL treats the RERUN clause as commentary.

MULTIPLE FILE TAPE Clause

The MULTIPLE FILE TAPE clause specifies the location of files on a multiple file reel. The MULTIPLE FILE TAPE clause is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

General format

MULTIPLE FILE TAPE CONTAINS {file-name-1 [POSITION integer-1]} …

General rules

1. The MULTIPLE FILE TAPE clause is required when more than one file shares the same physical reel of tape. Regardless of the number of files on a single reel, only those files that are used in the object program need be specified. If all file-names have been listed in consecutive order, the POSITION phrase need not be given. If any file in the sequence is not listed, the position relative to the beginning of the tape must be given. Not more than one file on the same tape reel may be open at one time.
2. Elastic COBOL treats the MULTIPLE FILE TAPE clause as commentary.

COMMITMENT Clause

The COMMITMENT CONTROL clause is AS/400 specific and not implemented; it gives a warning if used.

General format

COMMITMENT CONTROL FOR file-name-1

APPLY Clause

The APPLY clause applies attributes to files.

General format

APPLY LOCK-HOLDING {ON file-name-1}…

General rules

Each file-name-1 is set to LOCK MANUAL MULTIPLE.

12. Data Division

The data division describes the data that the object program is to accept as input, to manipulate, to create, or to produce as output. The data division is optional in a COBOL source program.

The following is the general format of the sections in the data division and defines the order of their presentation in the source program.

General format

```
[
DATA DIVISION.
        [
        CLASS-CONTROL.
        [repository-paragraph-contents]
        ]
        [
        FILE SECTION.
[[file-description-entry|sort-merge-file-description-entry
|78-level-description-entry|record-description-entry]…
[inline-file-control]]…
        ]
[
SHARED-STORAGE SECTION.
[78-level-description-entry|record-description-entry]…
        ]
        [
   WORKING-STORAGE SECTION.
   [78-level-description-entry|record-description-entry]…
        ]
        [
   LOCAL-STORAGE SECTION.
        [78-level-description-entry|record-description-entry]…
        ]
        [
        LINKAGE SECTION.
        [78-level-description-entry|record-description-entry]…
        ]
        [
        SCREEN SECTION.
        [78-level-description-entry|record-description-entry]…
        ]
]
```

Syntax rules

1. The CLASS-CONTROL, SHARED-STORAGE, LOCAL-STORAGE and SCREEN sections are not in ANSI-85 COBOL. LOCAL-STORAGE and SCREEN sections are present in the proposed COBOL-2002 standard.

General rules

1. The sections are created in a particular order. A section may validly only reference sections created prior. The order of creation is: SHARED-STORAGE, WORKING-STORAGE, FILE,
2. LOCAL, LINKAGE, SCREEN.
3. SCREEN SECTION items may refer to LINKAGE section items, but only those items passed by CONTENT rather than BY REFERENCE.
4. For deployment, it should be noted that each section generates its own Java inner class, named program_name$Wrk.class, program_name$Lnk.class, etc. These files must be deployed with the application as well as the main program_name.class.
5. The CLASS-CONTROL section is identical in content to the REPOSITORY paragraph in the ENVIRONMENT DIVISION and present only for increased compatibility with Micro Focus. Refer to the REPOSITORY paragraph for all information on CLASS-CONTROL.
6. SHARED-STORAGE contains data shared between all threads and all sessions of the COBOL program. It is created only once per runtime, regardless of the number of sessions and threads.
7. FILE contains the file definitions and record definitions for reading and writing files. Each session has its own copy of WORKING-STORAGE. Each program, thread, and recursion in a session has only one copy of WORKING-STORAGE. The WORKING-STORAGE is released only upon program termination or upon a CANCEL of the program.
8. WORKING-STORAGE contains the main working data for the program. Each session has its own copy of WORKING-STORAGE. Each program, thread, and recursion in a session has only one copy of WORKING-STORAGE. The WORKING-STORAGE is released only upon program termination or upon a CANCEL of the program.
9. LOCAL-STORAGE contains local working data for the program. Each session, thread and recursion has its own copy of LOCAL-STORAGE. The LOCAL-STORAGE is released upon program termination, GOBACK, or any activity which returns control to a higher level of control.
10. LINKAGE contains data passed into the program through the PROCEDURE DIVISION USING construct. Only data passed through the PROCEDURE DIVISION USING in this section is valid.
11. SCREEN contains data for displaying and accepting screens, textual or graphical. Elastic COBOL supports both textual and graphical screen sections.

Computer-independent data description

To make data as computer-independent as possible, the characteristics or properties of data are described in the data division in a standard data format expressed in terms of the appearance of graphic characters on a printed page of infinite breadth, rather than the manner in which data is stored internally in the computer or on a particular external medium. Numbers are described using the decimal number system, regardless of the radix and representation used by the computer; characters are described using characters from the COBOL character set.

NOTE - The term graphic character has the meaning defined in ISO/IEC 10646-1: 1993, and is essential to the definition of standard data format. A COBOL standard data format character corresponds to a graphic symbol only when the graphic symbol is the visual representation of a graphic character, and not the visual representation of a composite sequence.

Physical aspects of a file

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

1. The grouping of logical records within the physical limitations of the file medium.
2. The means by which the file shall be identified.

Characteristics of a file

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

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

A physical record is a physical unit of information transferred to or recorded on an output device or transferred from an input device. The size of a physical record is hardware dependent and bears no direct relationship to the size of the file of information contained on a device.

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

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

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

Record concepts

A record description consists of a set of data description entries that describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name, if required, followed by a series of independent clauses, as required.

Levels

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

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

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

Level-numbers

A system of level-numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, level-numbers for records start at 1. Less inclusive data items are assigned higher (not necessarily successive) level-numbers not greater in value than 49. There are special level-numbers, 66, 77, 78, and 88, that are exceptions to this rule. Separate entries are written in the source program for each level-number used.

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

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

1. Entries that specify elementary items or groups introduced by a RENAMES clause.
2. Entries that specify noncontiguous working storage and linkage data items.

3. Entries that specify condition-names.

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

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

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

Selection of character representation and radix

The value of a numeric item may be represented in either binary or decimal form, depending on the equipment. In addition there are several ways of expressing decimal. Since these representations are actually combinations of bits, they are commonly called binary-coded decimal forms. The selection of radix is generally dependent upon the arithmetic capability of the computer. If more than one arithmetic radix is provided, the selection is dependent upon the specification of the USAGE clause.

The selection of the means of representing character data is dependent on the capability of the computer and its external media and on factors included in the USAGE and PICTURE clauses. The method used in selecting the proper data form is also provided to allow the programmer to anticipate and/or control the selection.

Alphanumeric and national functions shall be represented in alphanumeric characters and national characters, respectively. The length of an alphanumeric or national function in standard data format characters is determined by the definition of the function. Integer and numeric functions may be used only in arithmetic expressions. An integer or numeric function represents the value resulting from the evaluation of the function without the restriction on composite of operands and/or receiving data items.

The size of an elementary data item or a group item shall be the number of digit positions or character positions in standard data format of the item. Synchronization and usage may cause a difference between this size and that required for internal representation.

Elastic COBOL uses two storage formats: internal and COBOL visible. Java objects are used internally to efficiently represent most datatypes. These objects attempt to use the most efficient form of data-manipulation for the data in question, and this internal form varies from operation to operation. There is no difference in the efficiency of operation between the various non-floating-point numeric types in operation; any difference in efficiency is related to conversion between internal form and COBOL visible form, which may vary from one JVM to another. Efficiency of floating-point numeric depends on the floating-point capabilities of the operating system and hardware, which may be different from the integer and fixed-point capabilities, and on the conversion related below.

Efficiency of data handling is related to conversion between internal and COBOL visible formats. This conversion is entirely automatic and irrelevant to COBOL functionality, but can be related to the efficiency of the program. This conversion

occurs whenever the same underlying data is referred to in a different form from last use. That is, if there is a group item G containing a numeric A and another numeric B redefining A, then the first access to A entails conversion; all subsequent accesses to A as a number require no conversion until accessed as either B or G. An access to B or G will require A to converted to COBOL visible form and then to G or B's internal form. A loop would be more efficient to do 100 references to A followed by 100 references to B rather than 100 interleaved references to A then B. Doing I/O generally requires conversion. The internal and COBOL visible form of non-numeric is generally identical, so no loss of efficiency in conversion occurs. This usage matches typical COBOL usage in inputting data, performing a number of operations on the data, and then outputting data; in this typical pattern, conversion occurs only once at the start of the run and once at the end of the run.

Limitations of character handling

Each coded character of the character sets supported by Elastic COBOL is processed at runtime as a single character position containing a single standard data format character except in cases where this specification clearly describes otherwise. If chosen as a computer's character set, ISO/IEC 10646-1 UTF-16 is supported by COBOL as a UCS-2 implementation.

NOTE - UTF-16 is supported as a Level U implementation as defined in ISO/IEC 10646-1.

Algebraic signs

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

The SIGN clause permits the programmer to state explicitly the location of the operational sign. This clause is optional; if it is not used, operational signs will be represented as TRAILING or as otherwise specified by the SPECIAL-NAMES paragraph.

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

Standard alignment rules

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

1. If the receiving data item is described as a fixed-point numeric item:
    a. The data is aligned by decimal point and is moved to the receiving digit positions with zero fill or truncation on either end as required.

b. When an assumed decimal point is not explicitly specified, the data item is treated as if it has an assumed decimal point immediately following its rightmost digit and is aligned as in rule 1a.

2. If the receiving data item is described as a floating-point numeric item, the alignment of the data is according to decimal point.
3. If the receiving data item is a fixed-point numeric-edited data item, 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 zeros.
4. If the receiving data item is alphabetic, alphanumeric, alphanumeric-edited, national, or national-edited, the sending data shall be moved, after any specified conversion, to the receiving character positions and aligned at the leftmost character position in the data item with space fill or truncation to the right, as required. If the JUSTIFIED clause is specified for the receiving item, alignment differs as specified in JUSTIFIED clause.

Item alignment for increased object-code efficiency

Some computer memories are organized in such a way that there are natural addressing boundaries in the computer memory (e.g., word boundaries, half-word boundaries, byte boundaries). The way in which data is stored is determined by the object program, and need not respect these natural boundaries.

The JVM has byte boundaries.

However, certain uses of data (e.g., in arithmetic operations or in subscripting) may be facilitated if the data is stored so as to be aligned on these natural boundaries. Specifically, additional machine operations in the object program may be required for the accessing and storage of data if portions of two or more data items appear between adjacent natural boundaries, or if certain natural boundaries bifurcate a single data item.

Data items that are aligned on these natural boundaries in such a way as to avoid such additional machine operations are defined to be synchronized.

Synchronization may be accomplished in two ways:

1. By use of the SYNCHRONIZED clause.
2. By recognizing the appropriate natural boundaries and organizing the data suitably without the use of the SYNCHRONIZED clause.

The SYNCHRONIZED clause is treated as commentary.

Explicit and implicit attributes

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

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

File section

The file section defines the structure of data, sort, and merge files. A file section is shared among recursive and thread calls. Each session has its own file section.

General format

FILE SECTION.

[{ file-description-entry| sort-merge-file-description-entry }

[78-level-description-entry|record-description-entry] …]

Syntax rules

1. The file section may be specified in a program definition.
2. In a file description entries there shall be at least one record description entry.

General Rules

A data-item format or table format VALUE clause specified in the file section is ignored. The initial value of a data item in the file section is undefined.

The initial value of a data item in the file section is defined to be the value given by the VALUE clause; if unspecified, the value or default value of the default-byte is used.

File description entry

In a COBOL program the file description entry (FD entry) represents the highest level of organization in the file section. The file section header is followed by a file description entry consisting of a level indicator (FD), a file-name, and a series of independent clauses. The entry itself is terminated by a period.

The file description entry furnishes information concerning the physical structure, identification, and the internal or external attributes of a file connector, of the associated data records, and of the associated data items. The file description entry also determines whether a file-name is a local name or a global name.

General format

Format 1 (sequential):

FD file-name-1

[IS EXTERNAL [BY literal-1]]

[IS GLOBAL]

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

[RECORD {CONTAINS integer-3 CHARACTERS}

| {IS VARYING IN SIZE [[FROM integer-4] [TO integer-5] CHARACTERS] [DEPENDING ON data-name-1]}

| {CONTAINS integer-6 TO integer-7 CHARACTERS}}]

[LINAGE IS {data-name-2|integer-8} LINES

[WITH FOOTING AT {data-name-3|integer-9}]
[LINES AT TOP {data-name-4|integer-10}]
[LINES AT BOTTOM {data-name-5|integer-11}]]
[CODE-SET IS alphabet-name-1]
[DATA {RECORD|RECORDS} {IS|ARE} identifier…]
[LABEL {RECORD|RECORDS} {IS|ARE} {STANDARD|OMITTED|identifier…}]
[BLOCK CONTAINS [integer TO] integer {RECORDS|CHARACTERS}]
[RECORDING MODE {{FIXED|F} | {VARIABLE|V}}]
[record-description-entry]…
[FILE-CONTROL inline-file-control .]

where inline-file-control-entry is:

assign-clause
|access-clause
|padding-clause
|reserve-clause
|record-key-clause
|file-status-clause
|organization-clause
|record-delimiter-clause
|recording-mode-clause
|alternate-record-key-clause
|password-clause
|control-area-clause
|data-size-clause
|index-size-clause
|nodisplay-clause
|pfkeys-clause
|cursor-column-clause
|file-locking-clause
|compression-clause
|encryption-clause
.

Format 2 (relative-or-indexed):

FD file-name-1
[IS EXTERNAL [BY literal-1]]
[IS GLOBAL]
[BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS|CHARACTERS}]
[RECORD {CONTAINS integer-3 CHARACTERS}
| {IS VARYING IN SIZE [[FROM integer-4] [TO integer-5] CHARACTERS]
[DEPENDING ON data-name-1]}
| {CONTAINS integer-6 TO integer-7 CHARACTERS}}]
[DATA {RECORD|RECORDS} {IS|ARE} identifier…]
[LABEL {RECORD|RECORDS} {IS|ARE} {STANDARD|OMITTED|identifier…}]
[BLOCK CONTAINS [integer TO] integer {RECORDS|CHARACTERS}]

[RECORDING MODE {{FIXED|F} | {VARIABLE|V}}]
[record-description-entry]…
[FILE-CONTROL inline-file-control .]

where inline-file-control-entry is:

assign-clause
|access-clause
|padding-clause
|reserve-clause
|record-key-clause
|file-status-clause
|organization-clause
|record-delimiter-clause
|recording-mode-clause
|alternate-record-key-clause
|password-clause
|control-area-clause
|data-size-clause
|index-size-clause
|nodisplay-clause
|pfkeys-clause
|cursor-column-clause
|file-locking-clause
|compression-clause
|encryption-clause
.

Syntax rules

ALL FORMATS

1. File-name-1 shall be specified in a file control entry.
2. The level indicator FD identifies the beginning of a file description entry and shall precede file-name-1.
3. The clauses that follow file-name-1 may appear in any order.
4. The DATA RECORD clause is treated as commentary.
5. The LABEL RECORD clause is treated as commentary.

Format 1 is the file description entry for a sequential file.

FORMAT 2

Format 2 is the file description entry for a relative file or an indexed file.

General rules

ALL FORMATS

A file description entry associates file-name-1 with a file connector.

An inline file-control-entry is applied to the file currently being defined (FD) just as if it had been applied in the FILE-CONTROL section in the ENVIRONMENT

DIVISION. All inline-file-control-entry clauses are identical to the corresponding ENVIRONMENT DIVISION clause and are documented in the ENVIRONMENT DIVISION chapter. The inline-file-control-entry clauses are available primarily to allow more file information to be defined in the same location, allowing more locality and allowing code generators targeting Elastic COBOL to generate file section code alongside the file's record definition.

FORMAT 1

If the file description entry for a sequential file contains the LINAGE clause and the EXTERNAL clause, the LINAGE-COUNTER data item is an external data item. If the file description entry for a sequential file contains the LINAGE clause and the GLOBAL clause, the special register LINAGE-COUNTER is a global name.

Sort-merge file description entry

The sort-merge file description entry furnishes information concerning the physical structure pertaining to a sort or merge file. The clauses of a sort-merge file description entry (SD entry) specify the size and the names of the data records associated with a sort file or a merge file. The rules for blocking and internal storage are peculiar to the SORT and MERGE statements. The sort-merge file description entry is terminated by a period.

General format

SD file-name-1

[RECORD {CONTAINS integer-3 CHARACTERS}

| {IS VARYING IN SIZE [[FROM integer-4] [TO integer-5] CHARACTERS]

[DATA {RECORD|RECORDS} {IS|ARE} identifier…]

Syntax rules

1. The level indicator SD identifies the beginning of the sort-merge file description entry and shall precede file-name-1.
2. The clauses that follow file-name-1 are optional, and their order of appearance is immaterial.
3. One or more record description entries shall follow the sort-merge file description entry.
4. File-name-1 or any record description entry associated with file-name-1 shall not be specified in an input-output statement other than following the word FROM or the word INTO.

General rules

The number of characters is specified in terms of alphanumeric character positions.

Working-storage section

The working-storage section is located in the data division of a program. Data described in the working-storage section is static data. The working-storage section describes records and subordinate data items that are not part of data files.

Data elements in working storage that bear a definite hierarchical relationship to one another shall be grouped into records according to the rules for formation of record descriptions. Data elements in the working-storage section that bear no hierarchical relationship to any other data item may be described as records that are single elementary items. All clauses that are used in record descriptions in the file section may be used in record descriptions in the working-storage section.

Items and constants in working storage that bear no hierarchical relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary items. Each of these items is defined in a separate data description entry that begins with the special level-number 77.

The initial value of a data item in the working-storage section that is not a based item may be specified by the data-item format or table format VALUE clause. Data items of category object reference and class pointer are initialized as indicated in the VALUE clause. Otherwise, the initial value of a data item is undefined. Otherwise, the initial value of a data item is defined by the value or default value of the default-byte compiler setting.

A data-item or table format VALUE clause specified in a record description entry containing the EXTERNAL clause takes effect only during the initial loading of a program or when the program is re-entered if its PROGRAM-ID specification contains the INITIAL attribute.

A working-storage is shared among recursive and thread calls. Each session has its own working-storage section.

General format

WORKING-STORAGE SECTION.

[77-level-description-entry|78-level-description-entry|record-description-entry]…

Local-storage section

The local-storage section is located in the data division of a program. Data described in the local-storage section is automatic data.

Data elements in local storage that bear a definite hierarchical relationship to one another shall be grouped into records according to the rules for formation of record descriptions. Data elements in the local-storage section that bear no hierarchical relationship to any other data item may be described as records that are single elementary items. All clauses that are used in record descriptions in the file section may be used in record descriptions in the local-storage section.

Items and constants in local storage that bear no hierarchical relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary

items. Each of these items is defined in a separate data description entry that begins with the special level-number 77.

The initial value of a data item in the local-storage section that is not a based item may be specified by the data-item format or table format VALUE clause. Data items of category object reference and class pointer are initialized as indicated in the VALUE clause. Otherwise, the initial value of a data item is undefined.

A local-storage is not shared among recursive, thread or session calls. Each logical view of the program has its own copy of the local-storage section.

General format

{LOCAL-STORAGE|BASED-STORAGE} SECTION.

[77-level-description-entry | record-description-entry | 78-level-description-entry] …

Linkage section

A linkage section that appears in a program definition describes data items that are referred to both by that element, when it is activated, and by the activating element. The mechanism by which a correspondence is established between the data items described in the linkage section and data items described in the activating element is described in 14.3, General rules of the procedure division. In the case of index-names, no such correspondence is established and index-names in the activated and activating source elements always refer to separate indices.

In a program definition, the linkage section is meaningful if and only if the program is to execute under the control of a CALL statement, and the CALL statement contains a USING phrase. If a data item in the linkage section is accessed in a program that is not a called program, the effect is undefined. If a program is activated by a run time entity that is other than a COBOL run time entity, access to linkage section items is meaningful if so designed by the calling entity.

A data-item format or table format VALUE clause specified in the linkage section is ignored. The initial value of a data item in the linkage section is undefined.

A linkage section is not shared among recursive, thread or session calls. Each CALL to a program has its own copy of the local-storage section. The data-items are not fully realized until the CALL links the data-items to the CALLing data items, as ordered by the PROCEDURE DIVISION USING parameters.

The set-up and tear-down time of the linkage section is comparatively large as it must be re-instantiated and re-linked for each CALL. Include only items actually referenced by the PROCEDURE DIVISION USING in the linkage section for the best performance. Passing items BY CONTENT is faster than passing items BY REFERENCE in Elastic COBOL.

General format

LINKAGE SECTION.

[77-level-description-entry | record-description-entry | 78-level-description-entry] …

Syntax rules

1. A data item defined in the linkage section of a source element may be referenced within the procedure division of that source element if, and only if, it satisfies one of the following conditions:
   a. It is an operand of the USING phrase of the procedure division header.
   b. It is subordinate to an operand of the USING phrase of the procedure division header.
   c. It is defined with a REDEFINES or RENAMES clause, the object of which satisfies the above conditions.
   d. It is subordinate to any item that satisfies the condition in rule 1c.
   e. It is a condition-name or index-name associated with a data item that satisfies one of the above conditions.

Shared-storage section

The shared-storage section is used for data that must be shared between separate sessions. Elastic COBOL has a session concept, allowing existing COBOL programs which are suitable for a multi-process environment to be executed in a multi-thread environment, where the session separation allows individual threads to behave as if they were separate processes. The shared-storage section allows data to be passed between these partitioned thread spaces.

The shared-storage section is allocated only once per runtime and is statically allocated. It cannot be successfully CANCELled.

General format

SHARED-STORAGE SECTION.

[77-level-description-entry|record-description-entry|78-level-description-entry]…

[In Data description entry, additional lines for General format 1:]

[IS SPECIAL-NAMES {CRT STATUS | SCREEN CONTROL | EVENT STATUS | CURSOR}]
[INDICATOR integer-1]
[GET-PROPERTY]
[SET-PROPERTY]
[LIKE like-identifier]
[FORMAT OF {DATE|TIME|TIMESTAMP} [IS format-literal] [SIZE IS size-integer]]
[EVENT
CLASS class-name-1
[ADD nonnumeric-literal-1]
[DELETE nonnumeric-literal-2]
[USING nonnumeric-literal-3]
[SET nonnumeric-literal-4]
[FOR nonnumeric-literal-5…]
END-EVENT]

Screen section

The screen section describes the screens to be displayed during terminal I-O. The screen section describes screen records and subordinate screen items.

General format

SCREEN SECTION.

[screen-description-entry | 78-level-description-entry] …

78-level description entry

A 78-level description entry is also known as a constant entry. A constant may be used in place of a literal.

General format

78 constant-name-1 VALUE literal-1.

Syntax rules

1. Constant-name-1 may be used anywhere that a format specifies a literal of the class and category of literal-1. In other than this entry, the effect is as if literal-1 were written where constant-name-1 is written.
2. Literal-1 shall not be a figurative constant.
3. Literal-1 shall not be a constant-name referencing a constant defined directly or indirectly as constant-name-1.

General rules

The class and category of constant-name-1 is the same as that of literal-1.

Record description entry

A record description consists of a set of data description entries that describe the characteristics of a particular record. Each data description entry consists of a level-number followed by the data-name or FILLER clause, if specified, followed by a series of independent clauses as required. A record description may have a hierarchical structure and therefore the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description and the elements allowed in a record description entry are explained in 13.2.4, Levels, and in Data description entry.

Data elements in the linkage section that bear a definite hierarchical relationship to one another shall be grouped into records according to the rules for formation of record descriptions. Data elements in the linkage section that bear no hierarchical relationship to any other data item may be described as records that are single elementary items.

77-level data description entry

Items in the linkage section, local-storage, and the working-storage section that bear no hierarchical relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary items. Each of these items is defined in a separate data description entry that begins with the special level-number 77. A data-name clause is required in each of these data description entries. Other clauses shall be used to complete the data description entry according to the rules for elementary data items.

Data description entry

A data description entry specifies the characteristics of a particular item of data. A level 1 data description entry within the file, local-storage, working-storage, local-storage, or linkage section determines whether the data record and its subordinate data items have local names or global names.

A level 1 data description entry in the working-storage section determines the internal or external attribute of the data record and its subordinate data items.

General format

Format 1 (data-description):

level-number {data-name-1|FILLER}
[REDEFINES data-name-2]
[IS EXTERNAL [BY literal-1]]
[IS IDENTIFIED BY literal-1]
[IS GLOBAL]
[{PICTURE|PIC} IS character-string]
[usage-clause]
[SIGN IS [LEADING|TRAILING] [SEPARATE CHARACTER]]
[occurs-clause]
[{SYNCHRONIZED|SYNC} [LEFT|RIGHT]]
[{JUSTIFIED|JUST} RIGHT]
[BLANK WHEN {ZERO | ZEROES | ZEROS }]
[VALUE IS literal-2]

.

where the following meta-language terms are described in the indicated paragraphs:

Term	Paragraph
occurs-clause	OCCURS clause
usage-clause	USAGE clause

Format 2 (renames):

66 data-name-1 RENAMES data-name-2 [{THROUGH|THRU} data-name-3].

Format 3 (condition-name):

88 condition-name-1 {VALUE|VALUES} [IS|ARE]
{ literal-1 [{THROUGH|THRU} literal-7] } …
[WHEN SET TO FALSE IS literal-8] .

Syntax rules

FORMAT 1

1. Level-number may be 77 or 1 through 49.
2. The data-name-1 or FILLER clause, if specified, shall immediately follow the level-number. The REDEFINES clause, if specified, shall immediately follow the data-name-1 or FILLER clause if either is specified; otherwise, it shall immediately follow the level-number. The remaining clauses may be written in any order.
3. The EXTERNAL clause shall be specified only in data description entries in the working-storage section whose level-number is 1.
4. The REDEFINES clause shall be specified in the same data description entry as the EXTERNAL clause.
5. The GLOBAL clause may be specified only in data description entries whose level-number is 1.
6. Data-name-1 shall be specified for any entry containing the GLOBAL or EXTERNAL clause, or for record descriptions associated with a file description entry that contains the EXTERNAL or GLOBAL clause.
7. The PICTURE clause shall not be specified for the subject of a RENAMES clause or for an item whose usage is index, object reference or pointer. For any other entry describing an elementary item, a PICTURE clause shall be specified except as indicated in syntax rule 8.
8. The PICTURE clause may be omitted when a VALUE clause with an alphanumeric literal or a national literal other than a figurative constant is specified. In this case, a PICTURE with the following characteristics is implied:

   a. If the literal is alphanumeric, 'PICTURE X(length)' is implied.

   b. If the literal is national, 'PICTURE N(length)' is implied.

   The term 'length' is the number of character positions in the literal specified in the VALUE clause.
9. The VALUE clause shall not be specified for data items of class index, object, or pointer.
10. The SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO clauses shall be specified only for an elementary data item.
11. If the USAGE clause is specified with the INTEGER or NUMERIC phrase, the only other clauses permitted are level-number, and data-name, screen-name, or FILLER.
12. The IDENTIFIED clause and EXTERNAL clause shall not be used for the same data item.

FORMATS 2 AND 3

The words THRU and THROUGH are equivalent.

FORMAT 3

Format 3 is used for each condition-name. Each condition-name requires a separate entry with level-number 88. The condition-name entries for a particular

conditional variable shall immediately follow the entry describing the item with which the condition-name is associated. A condition-name may be associated with any data description entry that contains a level-number except the following:

1. Another level 88 entry.
2. A level 66 entry.
3. A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than usage display) clauses.
4. A data item of the class index, object, or pointer.

General rules

1. Format 3 contains the name of the condition and the value, values, or range of values associated with the condition-name.
2. Multiple level 1 entries subordinate to any given level indicator represent implicit redefinition of the same area.

Screen description entry

Elastic COBOL supports two forms of screen section, textual and graphical.

Textual screen section support is for compatibility with existing screen section implementations, based on the X/Open or COBOL 2002 standard.

Graphical screen section support is for graphical element handling in a similar manner as the more traditional textual screen section. Graphical screen section support is intended for ready conversion of textual screens to graphical screens.

Textual screen section and graphical screen section elements may be mixed together on the same screen allowing for a more gradual approach to the creation of a graphical user interface. As such, a textual group item may have graphical item defined beneath it. So, while the following documents differences between the textual and graphical screen section, they are operating in a common framework and may interoperate. In all instances, graphical controls appear over textual information. (So, a label would obscure text at the same location.)

Both text and graphics use the same cell-base coordinate system, starting at column 1, line 1 in the upper-left corner of the screen/window; the final column and line is based on the size of the screen or window, but is generally column 80, line 25. (The DISPLAY STANDARD WINDOW is recommended as the first DISPLAY for graphical screen section programs; it controls the appearance and size of the main window in graphical programs.) Graphical elements may be placed at fractional coordinates (e.g., line 2.5, column 3.7); textual elements may only be placed at integer coordinates (line 2, column 3). Graphical elements may have fractional size (size 12.34, lines 23.45); textual elements have an integer size based on the size of their content.

In a very high-level manner, the LABEL graphical control represents an output textual field, and an ENTRY-FIELD graphical control represents an input textual field. The USING for input text is used for graphics input in an ENTRY-FIELD as well.

The Elastic COBOL graphical screen section is present on all graphical systems; it is not restricted to a single platform such as Windows. The graphical screen section support uses the more advanced JFC (Java Foundation Classes) graphical features of Java; for its use, Java 2 V1.3 or above is recommended.

This graphical screen section uses the same syntax developed for the textual screen section, and then enhances it with additional usage (HANDLE), additional clauses (property and style settings), and additional capabilities such as event handling. Existing verbs are extended (such as DISPLAY, ACCEPT, CLOSE, SHOW, HIDE), and new verbs have been added (such as MODIFY, INQUIRE, DESTROY).

Elastic COBOL supports multiple graphical formats, created for different uses. The graphical screen section is geared towards allowing textual screen sections to be brought forward into a graphical environment.

Elastic COBOL supports a number of internal copy files which contain constant definitions; these internal copy files are not really files, but rather a COPY statement is done to activate a certain set of constants to make usage of the graphical screen section simpler. The constant contents of these copy files is listed in the appendix. If a file by the same name is present, the normal COPY behavior will be observed, ignoring the internal copy file.

The graphical screen section is unusual in that it employs context-sensitive keywords. That is, certain keywords (mainly property and style names) are reserved only within the context of a particular identifier or particular handle. If a PUSH-BUTTON is the current handle being modified, then a PUSH-BUTTON's properties will be reserved; if not referring to a PUSH-BUTTON at the time, then the property names for a PUSH-BUTTON will be available for general use.

There is a set of handle-components which set the context; the graphical objects represented by these handle-components are also known as components, controls or widgets. These types are predefined by Elastic COBOL to refer to a certain graphical component. The current list of the handle-components is provided below. (There are certain other graphics capabilities, such as Windows, Message Boxes and Toolbars which do not appear at handle-components, but rather only through the DISPLAY verb.) See the appendix on handle-components for a complete list of available properties and styles. A handle-component may be expressed by name, such as RADIO-BUTTON, or by number using OBJECT object-number, such as OBJECT 5.

Handle-Component Table

Object		**Synonyms**	**Description**
Number	**Name**		
1	LABEL	TEXT-LABEL	Label for text, similar to a protected text field.
2	ENTRY-FIELD		Field where the user may enter data.
3	PUSH-BUTTON		Button which the user pushes.
4	CHECK-BOX		Button which the user checks on and off.
5	RADIO-BUTTON		Grouped button from which the user makes selection.
6	SCROLL-BAR		A scroll bar by which the user may select a value.
7	LIST-BOX		Box with a list of items.

Object		Synonyms	Description
Number	Name		
8	COMBO-BOX		Pull down list of items combined with entry-field.
9	FRAME	GROUP-FRAME	Graphical frame to grouping items visually for user.
10	TAB-CONTROL		Tabbed pane, allowing the user to select tabs.
11	BAR		Graphical bar for drawing.
12	GRID	GRID-CONTROL	Grid control, similar to a spreadsheet, for 2D data.
13	BITMAP		Image control, showing a graphical bitmap.
14	TREE-VIEW		A collapsible/expandable tree.
15	WEB-BROWSER		An HTML viewer.
1000	SLIDER		Similar to a scroll bar, but allows values to be made
1004	STATUS-BAR		Status bar at the bottom of a window.
1008	MENU		A graphical menu control.

Format 1 (Screen Group):

level-number {data-name-1 | FILLER }

[IS GLOBAL]

[AT|FROM] {LINE|ROW} [NUMBER IS [PLUS|+|-] {identifier-1|numeric-1}

[AT|FROM] {COLUMN|COL|POSITION|POS} [NUMBER IS [PLUS|+|-] {identifier-2|numeric-2}

[AT {integer-3 | identifier-3}]

[WITH AUTO | AUTO-SKIP | AUTOTERMINATE | AUTOMATIC]

[BACKGROUND-COLOR|BACKGROUND-COLOUR IS color-name]

[FOREGROUND-COLOR|FOREGROUND-COLOUR IS color-name]

[WITH [NO] BELL | BEEP]

[[NO] TAB]

[{ BLANK [SCREEN] } | { WITH ERASE [SCREEN] }]

[BLANK LINE]

[WITH ERASE {{EOL | TO END OF LINE} | {EOS | TO END OF SCREEN}}]

[WITH {BLINK | BLINKING}]

[LOWLIGHT|LOW|DIM|STANDARD]

[REVERSE-VIDEO|REVERSED|REVERSE]

[WITH {SECURE|NO-ECHO}]

[WITH SPACE-FILL]

[WITH ZERO-FILL]

[WITH SIZE {integer-4|identifier-4}]

[WITH [NO] PROMPT CHARACTER [identifier-5|literal-5]]

[WITH [NO] {HIGHLIGHT|BRIGHT|HIGH}]

[WITH {UNDERLINE|UNDERLINED}]

[WITH OVERLINE]

[WITH LEFTLINE]

[WITH GRIDLINE]

[WITH {REQUIRED|EMPTY-CHECK}]

[REDEFINES data-name-2]

[IS EXTERNAL [BY literal-1]]
[IS IDENTIFIED [BY literal-1]]
[PICTURE|PIC IS picture-string]
[usage-clause]
[SIGN IS [LEADING|TRAILING] [SEPARATE CHARACTER]]
[{SYNCHRONIZED|SYNC} {LEFT|RIGHT}]
[{JUSTIFIED|JUST} RIGHT]
[BLANK WHEN {ZERO|ZEROES|ZEROS}]
[VALUE IS literal-2]
[GET-PROPERTY]
[SET-PROPERTY]
Format 2 (Textual Elementary):
level-number {data-name-1 | FILLER }
[IS GLOBAL]
[AT|FROM] {LINE|ROW} [NUMBER IS [PLUS|+|-] {identifier-1|numeric-1}
[AT|FROM] {COLUMN|COL|POSITION|POS} [NUMBER IS [PLUS|+|-] {identifier-2| numeric-2}
[AT {integer-3 | identifier-3}]
[WITH AUTO | AUTO-SKIP | AUTOTERMINATE | AUTOMATIC]
[BACKGROUND-COLOR|BACKGROUND-COLOUR IS color-name]
[FOREGROUND-COLOR|FOREGROUND-COLOUR IS color-name]
[WITH [NO] BELL | BEEP]
[[NO] TAB]
[{ BLANK [SCREEN] } | { WITH ERASE [SCREEN] }]
[BLANK LINE]
[WITH ERASE {{EOL | TO END OF LINE} | {EOS | TO END OF SCREEN}}]
[WITH {BLINK | BLINKING}]
[LOWLIGHT|LOW|DIM|STANDARD]
[REVERSE-VIDEO|REVERSED|REVERSE]
[WITH {SECURE|NO-ECHO}]
[WITH SPACE-FILL]
[WITH ZERO-FILL]
[WITH SIZE {integer-4|identifier-4}]
[WITH [NO] PROMPT CHARACTER [identifier-5|literal-5]]
[WITH {FULL|LENGTH-CHECK}]
[WITH [NO] {HIGHLIGHT|BRIGHT|HIGH}]
[WITH {UNDERLINE|UNDERLINED}]
[WITH OVERLINE]
[WITH LEFTLINE]
[WITH GRIDLINE]
[WITH {REQUIRED|EMPTY-CHECK}]
[REDEFINES data-name-2]
[IS EXTERNAL [BY literal-1]]
[IS IDENTIFIED [BY literal-1]]
[PICTURE|PIC IS picture-string]

[usage-clause]
[SIGN IS [LEADING|TRAILING] [SEPARATE CHARACTER]]
[{SYNCHRONIZED|SYNC} {LEFT|RIGHT}]
[{JUSTIFIED|JUST} RIGHT]
[BLANK WHEN {ZERO|ZEROES|ZEROS}]
[VALUE IS literal-2]
[{FROM|SOURCE} identifier-8 | literal-1]
[VALUE identifier-8| literal-1]
[{TO|OBJECT} identifier-6]
[USING identifier-7]
[WITH {CONVERT | CONVERSION}]
[TRAILING-SIGN]
[STANDARD]
[WITH NUMERIC-FILL]
[CONTROL nonnumeric-literal-1|identifier-1]
BACKGROUND-HIGH
| BACKGROUND-LOW
| BACKGROUND-STANDARD]
[WITH COLOR {integer-1|identifier-1}]
[DEFAULT nonnumeric-literal-1]
[SCROLL {UP|DOWN} BY integer-2 {LINE|LINES}]
[UPPER]
[LOWER]
[ECHO]
[SAME]
[{OUTPUT JUSTIFIED {LEFT|RIGHT|CENTERED}}
| {OUTPUT JUSTIFIED {LEFT|RIGHT|CENTERED}}]
[BEFORE PROCEDURE {{entry-1 [THROUGH entry-2]} | NULLS | NULL}]
[AFTER PROCEDURE {{entry-1 [THROUGH entry-2]} | NULLS | NULL}]
[EXCEPTION PROCEDURE {{entry-1 [THROUGH entry-2]} | NULLS | NULL}]
[EVENT PROCEDURE {{entry-1 [THROUGH entry-2]} | NULLS | NULL}]
[HELP-ID [=] modify-value-1]
[ENABLED [=] modify-value-2]
[VISIBLE [=] modify-value-3]
[GET-PROPERTY]
[SET-PROPERTY]

Format 3 (Graphical Elementary):

level-number [data-name-1 | FILLER] {handle_component [title]}

title
[GRAPHICAL]
[CHARACTER]
[IDENTIFICATION|ID [IS|=] component-id]
[PICTURE|PIC IS picture-string]

[FROM [MULTIPLE|TABLE] from-item]

[VALUE [MULTIPLE|TABLE] from-item]

[TO [MULTIPLE|TABLE] to-item]

[USING [MULTIPLE|TABLE] using-item]

[[AT|FROM] {LINE|ROW} [NUMBER IS [PLUS|+|-] line-no
[CELL|CELLS|PIXEL|PIXELS]]

[[AT|FROM] {COLUMN|COL|POSITION|POS} [NUMBER IS [PLUS|+|-] col-no
[CELL|CELLS|PIXEL|PIXELS]]

[SIZE [IS|=] length [CELL|CELLS|PIXEL|PIXELS]]

[LINES [IS|=] height [CELL|CELLS|PIXEL|PIXELS]]

[CSIZE [IS|=] clength [CELL|CELLS]]

[CLINES [IS|=] cheight [CELL|CELLS]]

[TITLE [IS|=] title]

[KEY [IS|=] key-letter]

[STYLE [IS|=] style]

[FONT [IS|=] font-handle]

[COLOR|COLOUR IS color-val]

[WITH FOREGROUND-COLOR|FOREGROUND-COLOUR IS color-name]

[WITH BACKGROUND-COLOR|BACKGROUND-COLOUR IS color-name]

[WITH [NO] BELL | BEEP]

[HIGHLIGHT|HIGH|BOLD]

[LOWLIGHT|LOW|DIM|STANDARD|{NO {HIGHLIGHT|HIGH|BOLD}}]

[WITH REVERSE-VIDEO|REVERSED|REVERSE]

[WITH REQUIRED|EMPTY-CHECK]

[SYSTEM MENU]

[CONTROL FONT font-handle]

[CONTROL VALUE integer]

[BEFORE PROCEDURE IS { {entry-1 [THROUGH|THRU entry-2] } | NULL | NULLS }]

[AFTER PROCEDURE IS { {entry-1 [THROUGH|THRU entry-2] } | NULL | NULLS }]

[EXCEPTION PROCEDURE IS { {entry-1 [THROUGH|THRU entry-2] } | NULL | NULLS }]

[EVENT PROCEDURE IS { {entry-1 [THROUGH|THRU entry-2] } | NULL | NULLS }]

[AUTO]

[property-name [IS|=] {property-value | (property-value…) | {MULTIPLE|TABLE} property-table}]

[PROPERTY integer [IS|=] {property-value | (property-value…) | {MULTIPLE|TABLE} property-table }]

[property-name]

[{NOT|NO} property-name]

[GET-PROPERTY]

[SET-PROPERTY]

Graphical SCREEN SECTION Colors

Name	Value
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
YELLOW	6
WHITE	7
BRIGHT-BLACK	8
BRIGHT-BLUE	9
BRIGHT-GREEN	10
BRIGHT-CYAN	11
BRIGHT-RED	12
BRIGHT-MAGENTA	13
BRIGHT-BROWN	14
BRIGHT-YELLOW	14
BRIGHT-WHITE	15

Data Division Clauses

AT Clause

The AT clause positions the cursor at a specified line and column.

General format

AT {integer-1|identifier-1}

General rules

1. The AT clause specifies the line and column number on screen together in a manner similar to that of the separate LINE and COLUMN clauses.
2. Integer-1 or the value of identifier-1 is used as the master position value. This master position value must be 4 or 6 characters. If 4 characters, the first 2 digits are the line and the last 2 digits are the column; if 6 characters, the first 3 digits are the line and the last 3 digits are the column.
3. The separate LINE and COLUMN clauses are recommended over this clause for compatibility.

AUTO Clause

The AUTO clause causes the cursor to be automatically moved to the next field declared for the screen item during execution of an ACCEPT screen statement.

General format

WITH {AUTO | AUTO-SKIP | AUTOTERMINATE | AUTOMATIC}

General rules

1. An AUTO clause specified at the group level applies to each input screen item in that group.
2. The AUTO clause is ignored for an elementary output field.
3. The AUTO clause takes effect during the execution of an ACCEPT screen statement that references the screen item for which the AUTO clause is specified.
4. The AUTO clause causes the cursor to be automatically moved to the next field declared for the screen item when the last character of the input field whose definition contains this clause has data entered into it.
5. When the AUTO clause is specified for an input field that has no logical next field during input, then when that field is available for input during an ACCEPT screen statement and data is entered into the last character of the screen item, successful completion with normal termination of the ACCEPT statement results.

BACKGROUND-COLOR Clause

The BACKGROUND-COLOR clause specifies the background color for the screen item.

General format

WITH {BACKGROUND-COLOR | BACKGROUND-COLOUR} color-name

where color-name is:

{

{ 0 | BLACK } |
{ 1 | BLUE } |
{ 2 | GREEN } |
{ 3 | CYAN } |
{ 4 | RED } |
{ 5 | MAGENTA } |
{ 6 | BROWN | YELLOW } |
{ 7 | WHITE }

}

General rules

1. Color-name specifies the color number of the background color to be used for displaying the screen item.
2. A BACKGROUND-COLOR clause specified at the group level applies to each elementary screen item in that group.

BACKGROUND-Style Clause

General format

BACKGROUND-HIGH
| BACKGROUND-LOW
| BACKGROUND-STANDARD

Syntax rules

For textual screens, the BACKGROUND clauses are ignored.

BELL Clause

The BELL clause causes the terminal audio tone to sound.

General format

WITH [NO] {BELL | BEEP}

General rules

1. The use of this clause results in the audio tone sounding when the screen item in which it is specified is processed during the execution of a DISPLAY screen statement. The audio tone sounds once at the start of the display regardless of how many entries specify the clause.
2. A BELL clause specified at the group level applies to each elementary screen item in that group.
3. The NO option causes terminal audio tone to be turned off and not heard.

BLANK Clause

The BLANK clause clears a screen line or clears the whole screen before the screen item is displayed.

General format

{{WITH BLANK SCREEN} | {WITH ERASE SCREEN}} | {WITH BLANK LINE}

General rules

1. When the BLANK LINE clause is specified, the entire line specified for the screen item that is the subject of the entry, columns 1 through the end of the line, is cleared before the screen item is displayed.
2. When the BLANK SCREEN clause is specified, the screen is cleared and the cursor is placed at line 1, column 1 before the screen item is displayed. Upon clearing the screen, the background color for the entire screen is set to the value applicable at the time.
3. The BLANK SCREEN clause in combination with the BACKGROUND-COLOR clause for the same screen item, or for a screen item to which it is subordinate,

establishes the default background color to be used until the same combination is encountered specifying another background color.

4. The BLANK SCREEN clause in combination with the FOREGROUND-COLOR clause for the same screen item, or for a screen item to which it is subordinate, establishes the default foreground color to be used until the same combination is encountered specifying another foreground color.
5. The BLANK clause is ignored during execution of an ACCEPT screen statement.

BLANK WHEN ZERO Clause

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

General format

BLANK WHEN {ZERO | ZEROES | ZEROS }

Syntax rules

1. The BLANK WHEN ZERO clause shall be specified only for an elementary data item described with a PICTURE clause that specifies the category as numeric or numeric-edited.
2. The numeric or numeric-edited data description entry to which the BLANK WHEN ZERO clause is specified for a data item described with a PICTURE clause that specifies the category of the data item as numeric, the category of the data item is numeric-edited.

General rules

1. When the BLANK WHEN ZERO clause is used, the item will contain nothing but spaces if the value of the item is zero.
2. When the BLANK WHEN ZERO clause is used for an item whose PICTURE is numeric, the category of the item is considered to be numeric-edited.

BLINK Clause

The BLINK clause specifies that each character of the field blinks when it is displayed on the screen.

General Format

WITH {BLINK | BLINKING}

General Rules

1. A BLINK clause specified at the group level applies to each elementary screen item in that group.
2. When the BLINK clause is specified, the screen item will be displayed so that the characters that constitute the screen item will blink.

BLOCK CONTAINS Clause

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

General format

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

General rules

1. This clause is required except when one or more of the following conditions exist:
   a. A physical record contains one and only one complete logical record.
   b. The hardware device assigned to the file has one and only one physical record size.
   c. The number of records or characters contained in a block is specified in the operating environment.
2. The size of a physical record may be stated in terms of records unless one or more of the following situations exists, in which case the RECORDS phrase shall not be used:
   a. In mass storage files, where logical records may extend across physical records.
   b. The physical record contains padding (area not contained in a logical record).
   c. Logical records are grouped in such a manner that an inaccurate physical record size would be implied.

   Note: This clause is never required in Elastic COBOL
3. If the CHARACTERS phrase is specified, the physical record size is specified in terms of the number of alphanumeric character positions required to store the physical record, regardless of the types of characters used to represent the items within the physical record.
4. If integer-1 is not specified, integer-2 represents the exact size of the physical record. If integer-1 and integer-2 are both specified, they refer to the minimum and maximum size of the physical record, respectively.
5. If the associated file connector is an external file connector, all BLOCK CONTAINS clauses in the run unit that are associated with that file connector shall have the same value for integer-1 and integer-2.

CHARACTER Clause

Specify that the following clauses are for character, or text, mode systems only. Elastic COBOL uses only the GRAPHICAL route.

General format

CHARACTER

General rules

1. If an entry in a clause, every clause specified after CHARACTER is ignored. Elastic COBOL uses only graphical entries.
2. If a data-name replacement, then this item and all its children are ignored. Elastic COBOL uses only graphical entries.

CLINES Clause

Set the request number of lines for the screen item component on character systems. Elastic COBOL displays handle-components only on graphical systems, so this property is ignored.

General format

CLINES [IS|=] cheight [CELL|CELLS]

Syntax rules

CELL and CELLS are synonymous.

CODE-SET Clause

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

General format

CODE-SET IS alphabet-name-1

Syntax rules

1. Alphabet-name-1 shall reference an alphabet that defines an alphanumeric coded character set.
2. If there are record description entries associated with the file and no SELECT clauses are specified, alphabet-name-1 may be specified; and:
   a. if alphabet-name-1 is specified, all elementary data items of all record description entries associated with the file shall be described as usage display, and any signed numeric data items shall be described with the SIGN IS SEPARATE clause.

General rules

1. The CODE-SET clause identifies alphabets to be used for converting data from a coded character set on the storage medium to the native character set during input operations, and from the native character set to the coded character set on the storage medium during output operations.
2. Upon successful processing of an OPEN statement for the file referenced in this file description entry, the coded character set used to represent alphanumeric data on the storage medium is the one referenced by alphabet-name-1.

3. If the record description entries associated with the file do not contain a SELECT clause, the specified alphabet is used for code-set conversion of all data items in each record.
4. If record description entries associated with the file contain a SELECT clause, a record description entry is selected by evaluation of those SELECT clauses. If there are no record description entries associated with the file, the record description used for conversion is the description of the identifier or literal specified in the FROM phrase of a WRITE or REWRITE statement specifying the FILE option.

   Alphabet-name-1 is used for code-set conversion of each data item described with usage display in the selected record description entry.
5. For each data item to be converted:
   a. On input, each coded character from the storage medium is replaced with its associated native coded character as defined in the alphabet being used.
   b. On output, each native coded character in the record is replaced for the storage medium with its associated coded character as defined in the alphabet being used.
6. If the CODE-SET clause is not specified, the native character set is assumed for data on the external media.
7. If the associated file connector is an external file connector, all CODE-SET clauses in the run unit that are associated with that file connector shall have the same character set.

COLOR Clause

Specify a combined color for foreground and background colors. Unlike foreground-color and background-color, the color clause does not require an integer, but instead allows an identifier.

General format

WITH COLOR {integer-1|identifier-1}

General rules

Select a foreground color, background color, and attributes; add the corresponding values together to form the desired color value.

Color	Foreground	Background
Black	1	32
Blue	2	64
Green	3	96
Cyan	4	128
Red	5	160
Magenta	6	192
Brown	7	224
White	8	256

Attributes

Reverse-Video	1024
Low-Intensity	2048

High-Intensity	4096
Underline	8192
Blinking	16384
Protected	32768
Background-Low	65536
Background-High	131072

COLUMN Clause

The COLUMN clause specifies the horizontal screen coordinate for a screen item.

General format

[AT|FROM]
{COLUMN|COL|POSITION} NUMBER [PLUS|+] {identifier-1|integer-1} [CELL|CELLS|PIXEL|PIXELS]

Syntax rules

1. COLUMN, COL, and POSITION are synonyms.
2. PLUS and + are synonyms.
3. Identifier-1 shall be described in the file, working-storage, or linkage section as an elementary unsigned integer data item.
4. The PLUS phrase shall not be specified for the first elementary item in a screen record.
5. CELL and CELLS are synonymous. It applies only to graphical screen items.

General rules

1. The COLUMN clause specifies the column in which the leftmost character of the screen item is to appear on the screen during the execution of an ACCEPT screen or a DISPLAY screen statement. Positioning of the screen record and within the screen record appears the same on the terminal display regardless of whether the whole screen record or just a portion of it is referenced in an ACCEPT screen or a DISPLAY screen statement.
2. If the COLUMN clause does not specify PLUS, the clause gives the column number relative to the first column of the screen record. A column number of 1 represents the first column of the screen record.
3. If the PLUS phrase is specified in the COLUMN clause, the column number is relative to the end of the preceding screen item in the same screen record, such that if COLUMN PLUS 1 is specified, the screen item starts immediately following the preceding screen item. PLUS denotes a column position that is increased by the value of identifier-1 or integer-3.
4. A setting of COLUMN 0 is assumed for screen descriptions that specify the LINE clause but omit the COLUMN clause.
5. If both the LINE clause and the COLUMN clause are omitted, the following apply:
    a. if no previous screen item has been defined, LINE 1 COLUMN 1 of the screen is assumed.

b. if a previous screen item has been defined, the line of that previous item and COLUMN PLUS 1 is assumed.

6. If a column number of zero is specified, the current column position is assumed.
7. CELL|CELLS is redundant; the default positioning is based on cells.
8. PIXEL|PIXELS specifies the value in terms of pixels directly allowing directly control over placement. Avoid direct pixel control where possible.

CONTROL Clause

The control clause allows dynamic modification of screen attributes. Elastic COBOL supports the extended SET syntax instead, so this clause is syntax checked only.

General format

CONTROL nonnumeric-literal-1|identifier-1

Syntax rules

1. The CONTROL format is currently syntax checked, but not supported.
2. For runtime modification of textual screen settings, see the SET statement for the COBOL 2002 settings.

CONTROL FONT Clause

Set the control font property of a graphical component.

General format

CONTROL FONT font-handle

CONTROL VALUE Clause

Set the control value property of a graphical component.

General format

CONTROL VALUE integer

CONVERT Clause

The convert clause allows non-DISPLAY items to be displayed and accepted. Elastic COBOL always supports this, so this clause is syntax-checked only.

General format

WITH {CONVERT | CONVERSION}

Syntax rules

The CONVERT clause is syntax-checked only.

CSIZE Clause

Set the request number of columns for the screen item component on character systems. Elastic COBOL displays handle-components only on graphical systems, so this property is ignored.

General format

CSIZE [IS|=] clength [CELL|CELLS]

Syntax rules

CELL and CELLS are synonymous.

Data-name, screen-name, or FILLER Clause

A data-name or screen-name specifies the name of the item being described. The key word FILLER may be used to specify an item that is not referenced explicitly.

General format

[data-name-1 | screen-name-1 | FILLER]

Syntax rules

Data-name-1, screen-name-1, or the key word FILLER, if specified, shall be the first word following the level-number in each data description entry.

General rules

1. If data-name-1, screen-name-1, and FILLER are omitted, the data or screen item being described is treated as though FILLER had been specified.
2. If data-name-1, screen-name-1, and FILLER are omitted, the data or screen item The word FILLER may be used to name a data or screen item. Under no circumstances shall a FILLER item be referred to explicitly. However, the key word FILLER may be used to name a conditional variable because such use does not require explicit reference to the data item itself, but only to the condition-name associated with the value contained therein.

DEFAULT Clause

Specify a default value to be displayed in an input field for editing by the user. The DEFAULT clause is syntax-checked only for textual screen items.

General format

DEFAULT nonnumeric-literal-1

UPDATE

Syntax rules

1. For textual screens, the DEFAULT clause is ignored.
2. The UPDATE clause is the DEFAULT clause with no value specified.

ECHO Clause

The ECHO clause is syntax-checked only for textual screen items.

General format

ECHO

Syntax rules

For textual screens, the ECHO clause is ignored.

ENABLED Clause

Set the enabled status for a graphical component.

General format

ENABLED [=] value

Syntax rules

ENABLED is ignored for textual screens.

General rules

ENABLED determines if a graphical component is enabled for user interaction. Non-enabled components are typically displayed in a different manner.

ERASE Clause

The ERASE clause clears part of the line or the screen starting at the cursor position.

General format

WITH ERASE [{EOL|TO END OF LINE} | {EOS|TO END OF SCREEN} | SCREEN]

Syntax rules

1. The word EOL is equivalent to the words TO END OF LINE.
2. The word EOS is equivalent to the words TO END OF SCREEN

General rules

1. When the ERASE clause is specified, a portion of the screen is cleared before the screen item is displayed. The clearing begins at the line and column coordinates specified for the subject of the entry and continues as follows:
   - If LINE is specified, the clearing continues to the end of the line.
   - If SCREEN is specified, the clearing continues to the end of the screen.
2. The ERASE clause is ignored during execution of an ACCEPT screen statement.

3. If neither TO END OF LINE nor TO END OF SCREEN is specified, the entire screen is cleared.

EVENT Clause

This event clause is used by Elastic COBOL style graphics for generation of an event adapter class.

General format

```
EVENT
        CLASS class-name-1
                [ADD nonnumeric-literal-1]
                [DELETE nonnumeric-literal-2]
                [USING nonnumeric-literal-3]
                [SET nonnumeric-literal-4]
                [FOR nonnumeric-literal-5…]
END-EVENT
```

EXTERNAL Clause

The EXTERNAL clause specifies that a data item or a file connector is external. The constituent data items and group data items of an external data record are available to every run time entity in the run unit that describes that record.

General format

```
IS EXTERNAL [BY literal-1]
```

Syntax rules

1. The EXTERNAL clause may be specified only in file description entries or in record description entries in the working-storage section.
2. In the same source element, the name that is externalized for the subject of the entry whose level-number is 1 that includes the EXTERNAL clause shall not be the same name that is externalized for any other data description entry that includes the EXTERNAL clause.
3. Literal-1 shall be an alphanumeric literal and shall not be a figurative constant.

General rules

1. If the EXTERNAL clause is specified in a record description entry, the data contained in the record is external and may be accessed and processed by any run time entity in the run unit that describes it subject to the following general rules.
2. Literal-1, if specified, is the name of the file or record that is externalized to the operating environment. If literal-1 is not specified, the externalized name of the file or record is the name specified in the file description entry or the data name clause, respectively.

3. Within a run unit, if two or more source elements describe the same external data record, each name that is externalized to the operating environment for the record description entries shall be the same; the VALUE clause specification, if any, for each record name of the associated record description entries shall be identical; and the records shall define the same number of standard data format characters. However, a source element that describes an external record may 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 source elements in the run unit.
4. Use of the EXTERNAL clause does not imply that the associated file-name or data-name is a global name.
5. The file connector associated with this description entry is an external file connector.

FONT Clause

Set the font property of a graphical component to a given font handle.

General format

FONT [IS|=] font-handle

FOREGROUND-COLOR Clause

The FOREGROUND clause specifies the foreground color for the screen item.

General format

WITH {FOREGROUND-COLOR | FOREGROUND-COLOUR} color-name

where color-name is:

```
{
        { 0     | BLACK } |
        { 1     | BLUE } |
        { 2     | GREEN } |
        { 3     | CYAN } |
        { 4     | RED } |
        { 5     | MAGENTA } |
        { 6     | BROWN | YELLOW } |
        { 7     | WHITE }
}
```

General rules

1. Color-name specifies the color number of the foreground color to be used for displaying the screen item. The color associated with the color number is specified in Color number.
2. A FOREGROUND-COLOR clause specified at the group level applies to each elementary screen item in that group.

FORMAT Clause

The format clause specifies the format of a date/time/timestamp data item.

General format

FORMAT OF {DATE|TIME|TIMESTAMP} [IS format-literal] [SIZE IS size-integer]

General rules

Syntax rules

1. format-literal must be a nonnumeric-literal expressing a valid date/time/timestamp format; the literal must include only date related items for DATE, time related items for TIME, or any valid combination for TIMESTAMP. See appendix for full information on date/time escapes.
2. If FORMAT is used, it replaces PICTURE in defining the item.
3. The default format-literal for DATE is @Y-%m-%d.
4. The default format-literal for TIME is %H:%M:%S.
5. The default format-literal for TIMESTAMP is @Y-%m-%d-%H.%M.%S.@Sm.

General rules

1. A default PICTURE is created from the FORMAT; the default picture is always PIC X(n), where n is the number of bytes calculated for the format literal. This default picture may have its size altered by the SIZE clause.
2. Any identifier with a FORMAT clause may be used for SQL date/time/timestamp storage and retrieval.

Certain intrinsic functions take or use only FORMAT data items.

FROM Clause

The FROM clause specifies the source of data for DISPLAY screen statement.

General format

{FROM|SOURCE} {identifier-1|literal-1}

Syntax rules

1. Identifier-1 shall be defined in the file, working-storage, local-storage, or linkage sections.
2. The category of identifier-1 and literal-1 shall be a permissible category as a sending item in a MOVE statement where the receiving item has the same PICTURE clause as the subject of the entry.

General rules

The subject of the entry is an output screen item.

FROM Clause Graphical

The FROM clause specifies the source of data for the DISPLAY of a screen item.

General format

FROM [MULTIPLE|TABLE] identifier-1|literal-1

Syntax rules

MULTIPLE and TABLE are synonymous.

General rules

1. Identifier-1 or literal-1 is the source of the data; this data is read and used for the display of the input item during a DISPLAY.
2. If identifier-1 is a table and the component supports the MULTIPLE type, then MULTIPLE identifier-1 indicates that each element of the table identifier-1 shall be used to fill the component.

FULL Clause

The FULL clause specifies that the operator shall either leave the screen item completely empty or fill it entirely with data.

General format

WITH {FULL|LENGTH-CHECK}

General rules

1. If a FULL clause is specified in a group screen item, it applies to each elementary input screen item in that group that does not have the JUSTIFIED clause specified.
2. The FULL clause is effective during the execution of any ACCEPT screen statement that causes the screen item to be accepted, provided the cursor enters the screen item at some time during the execution of the ACCEPT statement.
3. If the screen item is alphanumeric, then to satisfy this clause, either the entire item shall contain spaces, or both the first and last character positions shall contain non-space characters.
4. If the screen item is numeric or numeric edited, then to satisfy the clause either the value shall be zero or there shall be no digit position in which zero suppression has taken effect.
5. For fields that are both input and output, the FULL clause may be satisfied by the contents of the literal or identifier referenced in the FROM or USING clause, as well as operator keyed data.
6. Until the FULL clause is satisfied, the normal termination key is rejected.

7. The FULL clause shall not be effective if a function key is used to terminate the execution of the ACCEPT statement.
8. The specification of the FULL and REQUIRED clauses together requires that the field be entirely filled before the normal termination key has any effect.
9. The FULL clause is ignored for an elementary output field.

GET-PROPERTY Clause

The GET-PROPERTY clause generates two get methods for the identifier, allowing its value to be retrieved as a Java Bean property by an external Elastic COBOL or Java program. This clause is suitable for enhanced Elastic COBOL/Java integration.

General format

GET-PROPERTY

Syntax rules

1. Only one GET-PROPERY may be specified per identifier.
2. The GET-PROPERTY may be specified in a PROGRAM-ID style program.
3. A single identifier may have both a GET-PROPERTY and a SET-PROPERTY.

General rules

1. Two get methods are generated for each identifier with a GET-PROPERTY.
2. If the COBOL variable must be subscripted for access, then integer subscripts, one per dimension of the table, are generated in the Java method signature; all indexes are based at one.
3. The first method is getName() which returns a human-readable String (generated as if by DISPLAY) for most types or Objects for object references, the second is getNameAsType() which returns a Java type natural to the COBOL type. The first letter of name is capitalized, as is any letter after a '-'; any '-' is removed from the name. The type is given by the following table.

COBOL Type to Java Method Reference

COBOL Type	setName(String)	getName([int dimension[,dimension]...]type)
Display	String	java.math.BigDecimal
Comp-D	String	java.math.BigDecimal
Packed-Decimal	String	java.math.BigDecimal
Binary	String	byte (1 byte), short (2 bytes), int (3-4 bytes), long (5-8 bytes)
Binary-Rev	String	byte (1 byte), short (2 bytes), int (3-4 bytes), long (5-8 bytes)
Comp-X	String	byte (1 byte), short (2 bytes), int (3-4 bytes), long (5-8 bytes)
Comp-X-Rev	String	byte (1 byte), short (2 bytes), int (3-4 bytes), long (5-8 bytes)
Comp-S	String	short
Comp-1	String	float
Comp-1-MVS	String	float
Comp-1-REV	String	float

COBOL Type	setName(String)	getName([int dimension[,dimension]…]type)
Comp-2	String	double
Comp-2-MVS	String	double
Comp-2-REV	String	double
Numeric-Edited	String	String
Alphanumeric	String	String
Alphanumeric-Edited	String	String
Alphabetic	String	String
National	String	String
National-Edited	String	String
Index	int	int
Object Reference	Object Class	Object Class
Numeric-Hashtable	Hashtable	java.util.Hashtable
Alphanumeric-Hashtable	Hashtable	java.util.Hashtable
Object-Hashtable	Hashtable	java.util.Hashtable
Jbyte	byte	byte
Jshort	short	short
Jint	int	int
Jlong	long	long
Jfloat	float	float
Jdouble	double	double
Jboolean	Boolean	Boolean
Handle	Object	Object

Example:

01 alpha pic 999 get-property.

01 beta pic 999 binary value 0 get-property.

01 gamma pic 999 comp-1 get-property.

01 delta pic 999 comp-2 get-property.

01 my-string pic x(10) get-property.

01 my-object object reference "java.util.Vector" get-property.

Generates methods:

public String getAlpha()

public java.math.BigDecimal getAlphaAsBigDecimal()

public String getBeta()

public short getBetaAsShort()

public String getGamma()

public float getGammaAsFloat()

public String getDelta()

public double getDeltaAsDouble()

public String getMyString()

public String getMyStringAsString()

public java.util.Vector getMyObject()

public java.util.Vector getMyObjectAsVector()

GLOBAL Clause

The GLOBAL clause specifies that a constant-name, a data-name, a file-name, a report-name, or a screen-name is a global name. A global name is available to every program contained within the program that declares it.

General format

IS GLOBAL

Syntax rules

1. The GLOBAL clause may be specified only in the following entries:
   a. A data description entry where the level-number is 1 that is specified in the file, working-storage, local-storage, or linkage section.
   b. A screen-description entry where the level-number is 1.
   c. A file description entry.
2. In the same data division, the description entries for any two items for which the same name is specified shall not include the GLOBAL clause.
3. If the SAME RECORD AREA clause is specified for several files, the record description entries or the file description entries for these files shall not include the GLOBAL clause.
4. If the GLOBAL clause is not specified in the file description entry of a containing program, the file shall not be referenced directly or indirectly by any input-output statements in any contained program.
5. The GLOBAL clause shall not be specified in a factory definition, an object definition, or a method definition.

General rules

1. A data-name, file-name, or screen-name described using a GLOBAL clause is a global name. All data-names subordinate to a global name are global names. All condition-names associated with a global name are global names.
2. A statement in a program contained directly or indirectly within a program that describes a global name may reference that name without describing it again. (See Scope of names.)
3. If the GLOBAL clause is used in a data description entry that contains the REDEFINES clause, it is only the subject of that REDEFINES clause that possesses the global attribute.

GRAPHICAL Clause

Specify that the following clauses are for graphical systems only. Elastic COBOL uses only the graphical route.

General format

GRAPHICAL

General rules

1. If an entry in a clause, the GRAPHICAL clause is ignored, but subsequent clauses are used. Elastic COBOL uses only graphical entries.
2. If a data-name replacement, then this item and all its children are used. Elastic COBOL uses only graphical entries.

GRIDLINE Clause

The GRIDLINE clause specifies that each character of the field is gridlined, rendered with surrounding lines, when it is displayed on the screen.

General format

WITH GRIDLINE

General rules

1. If the GRIDLINE clause is specified at group level, it applies to each elementary screen item in that group.
2. When the GRIDLINE clause is specified, the screen item will be displayed so that the characters that constitute the screen item are gridlined.
3. Micro Focus uses the GRID keyword; in Elastic COBOL this is left open for use by the AWT 1.0 GUI, but the GRID keyword may be substituted for GRIDLINE by using the –words compiler option.

HELP-ID Clause

Set the help-id topic number for a graphical component.

General format

HELP-ID [=] value

Syntax rules

HELP-ID is ignored for textual screens.

General rules

HELP-ID specifies a help identification number to a graphical component, allowing any help system in place to determine the help topic number.

HIGHLIGHT Clause

The HIGHLIGHT clause specifies that the field is to appear on the screen with the highest level of intensity.

General format

WITH [NO] {HIGHLIGHT|BRIGHT|HIGH}

General rules

1. A HIGHLIGHT clause specified at the group level applies to each elementary screen item in that group.
2. When the HIGHLIGHT clause is specified, the characters that constitute the screen item will be displayed in the foreground color at the highest intensity.
3. For NO HIGHLIGHT, see LOWLIGHT.

IDENTIFICATION = Clause

Specify an identification number for a graphical component. This clause is ignored for textual screen items.

General format

IDENTIFICATION|ID [IS|=] component-id

IDENTIFIED Clause

The IDENTIFIED clause specifies an additional externally available identifier used by the EXTERNAL-FORM usage for purposes such as Servlets and CGI.

General format

IS IDENTIFIED BY literal-1

Syntax rules

Literal-1 may be any nonnumeric-literal suitable for the external-form of which this data item forms a part.

General rules

1. The IDENTIFIED BY is used to specify the externally available identification for EXTERNAL-FORM items. The IDENTIFIED BY may be used on any alphanumeric class or numeric class or external-form identifier that is not defined as EXTERNAL. It may include spaces where appropriate.
2. Typically, the IDENTIFIED BY is used by CGI programs to specify the external form variable to which the variable is attached.

IDENTIFIED BY Clause

Specify an alternative name to the runtime. This is useful primarily for specifying an HTML template file for EXTERNAL-FORM usage.

General format

IS IDENTIFIED [BY literal-1]

Syntax rules

IDENTIFIED may not be used in the same definition as EXTERNAL.

General rules

The variable name known to the runtime will be literal-1.

INDICATOR Clause

The INDICATOR clause specifies that the identifier is an INDICATOR or start of an INDICATOR area. This syntax is AS/400 related and currently accepted as syntax only; the runtime currently ignores the information.

General format

INDICATOR integer-1

Syntax rules

integer-1 shall be a value from 1 through 99.

General rules

1. INDICATOR specifies the AS/400 DDS indicator number of start of indicator area when done with an OCCURS.
2. INDICATOR currently has no meaning outside of AS/400 DDS.

JUSTIFIED Clause

The JUSTIFIED clause specifies right justification of data within a receiving data item or screen item.

General format

{JUSTIFIED|JUST} RIGHT

Syntax rules

1. The JUSTIFIED clause shall be specified only at the elementary item level.
2. JUST is an abbreviation for JUSTIFIED.
3. The JUSTIFIED clause shall be specified only for a data item whose category is alphabetic, alphanumeric, Boolean, or national.

General rules

1. When the receiving data item is described with the JUSTIFIED clause and the sending data item is larger than the receiving data item, the leftmost characters positions or Boolean positions of the sending item shall be truncated.
2. When the receiving data item is described with the JUSTIFIED clause and it is larger than the sending data item, the data is aligned at the rightmost character position in the data item with zero fill for the leftmost Boolean positions and space fill for the leftmost character positions. For data items implicitly or explicitly described as usage national, national zeros shall be used for zero fill

and national SPACES shall be used for space fill. For data items implicitly or explicitly described as usage display, alphanumeric zeros shall be used for zero fill and alphanumeric SPACES shall be used for space fill. For data items implicitly or explicitly described as usage bit, bit zeros shall be used for zero fill.

3. When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply (see Standard alignment rules).

KEY Clause

Set the key letter property of a graphical component. The key-letter is the hotkey, used to transfer focus to the given component using the keyboard.

General format

KEY [IS|=] key-letter

LEFTLINE Clause

The LEFTLINE clause specifies that each character of the field is leftlined, rendered with a line on the left side, when it is displayed on the screen.

General format

WITH LEFTLINE

General rules

1. If the LEFTLINE clause is specified at group level, it applies to each elementary screen item in that group.
2. When the LEFTLINE clause is specified, the screen item will be displayed so that the characters that constitute the screen item are leftlined.

Level-number Clause

The level-number indicates the position of a data item or screen item within the hierarchical structure of a logical record or a report group. In addition, it is used to identify entries for working storage items, local-storage items, linkage items, condition-names, and the RENAMES clause.

General format

level-number

Syntax rules

1. A level-number is required as the first element in each data description or screen description entry.
2. Data description entries subordinate to a FD or SD entry shall have level-numbers with the values 66, 88,or 1 through 49.
3. A level-number in the range of 1 through 9 may be specified as 01 through 09.

4. Data description entries in the working-storage section, local-storage section, and linkage section shall have level-numbers 66, 77, 88, or 1 through 49.
5. Screen description entries shall have level-numbers 1 through 49.

General rules

1. The level-number 1 identifies the first entry in each record description or report group.
2. Special level-numbers have been assigned to certain entries where there is no real concept of hierarchy:
   a. Level-number 77 is assigned to identify noncontiguous working storage data items, noncontiguous linkage data items, and shall be used only as described by the data description format of the data description entry.
   b. Level-number 66 is assigned to identify RENAMES entries and shall be used only as described by the renames format of the data description entry.
   c. Level-number 88 is assigned to entries that define condition-names associated with a conditional variable and to define criteria to be used to validate a data item. Level-number 88 shall be used only as described by the condition-name format or the validation format of the data description entry.
   d. Level-number 78 is assigned to constant entries.
3. Multiple level 1 entries subordinate to any given level indicator represent implicit redefinition of the same area.

LIKE Clause

This format defines the identifier like another identifier.

General format

LIKE identifier-1

Syntax rules

Identifier-1 must have already been defined.

General rules

The picture, usage, type, classname, sign position and format of identifier-1 are used for the current identifier. That is, the current variable is defined as if it were the same as identifier-1.

LINAGE Clause

The LINAGE clause provides a means for specifying the depth of a logical page in terms of number of lines. It also provides for specifying the size of the top and bottom margins on the logical page, and the line number, within the page body, at which the footing area begins.

General format

[LINAGE IS {data-name-1|integer-1} LINES
[WITH FOOTING AT {data-name-2|integer-2}]
[LINES AT TOP {data-name-3|integer-3}]
[LINES AT BOTTOM {data-name-4|integer-4}]]

Syntax rules

1. Data-name-1, data-name-2, data-name-3, data-name-4 shall reference elementary unsigned numeric integer data items.
2. Data-name-1, data-name-2, data-name-3, data-name-4 may be qualified.
3. Integer-2 shall not be greater than integer-1.
4. Integer-3, integer-4 may be zero.

General rules

1. The LINAGE clause provides a means for specifying the size of a logical page in terms of number of lines. The logical page size is the sum of the values referenced by each phrase except the FOOTING phrase. If the LINES AT TOP or LINES AT BOTTOM phrases are not specified, the values of these items are zero. If the FOOTING phrase is not specified, no end-of-page condition independent of the page overflow condition exists.

   There is not necessarily any relationship between the size of the logical page and the size of a physical page.
2. Integer-1 or the value of the data item referenced by data-name-1 specifies the number of lines that may be written and/or spaced on the logical page. The value shall be greater than zero. That part of the logical page in which these lines may be written and/or spaced is called the page body.
3. Integer-2 or the value of the data item referenced by data-name-2 specifies the line number within the page body at which the footing area begins. The value shall be greater than zero and not greater than integer-1 or the value of the data item referenced by data-name-1.

   The footing area is the area of the page body between the line represented by integer-2 or the value of the data item referenced by data-name-2 and the line represented by integer-1 or the value of the data item referenced by data-name-1, inclusive.
4. Integer-3 or the value of the data item referenced by data-name-3 specifies the number of lines in the top margin on the logical page. The value may be zero.
5. Integer-4 or the value of the data item referenced by data-name-4 specifies the number of lines in the bottom margin on the logical page. The value may be zero.
6. Integer-1, integer-3, and integer-4, if specified, are used at the time the file is opened by the execution of an OPEN statement with the OUTPUT phrase, to specify the number of lines in each of the indicated sections of a logical page. Integer-2, if specified, is used at that time to define the footing area. These values are used for all logical pages written for that file during a given execution of the program.

7. The values of the data items referenced by data-name-1, data-name-3, and data-name-4, if specified, are used as follows:

    a. The values of the data items, at the time an OPEN statement with the OUTPUT phrase is executed for the file, are used to specify the number of lines that are in each of the indicated sections for the first logical page.

    b. The values of the data items, at the time a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs are used to specify the number of lines that are in each of the indicated sections for the next logical page.

8. The value of the data item referenced by data-name-2, if specified, at the time an OPEN statement with the OUTPUT phrase is executed for the file, is used to define the footing area for the first logical page. At the time a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs, it is used to define the footing area for the next logical page.

9. A LINAGE-COUNTER is generated by the presence of a LINAGE clause. The value in the LINAGE-COUNTER at any given time represents the line number at which the device is positioned within the current page body. The rules governing the LINAGE-COUNTER are as follows:

    a. A separate LINAGE-COUNTER is supplied for each file described in the file section whose file description entry contains a LINAGE clause.

    b. LINAGE-COUNTER may be referenced only in procedure division statements; however only the input-output control system may change the value of LINAGE-COUNTER. Since more than one LINAGE-COUNTER may exist in a program, the user shall qualify LINAGE-COUNTER by file-name when necessary.

    c. LINAGE-COUNTER is automatically modified, according to the following rules, during the execution of a WRITE statement to an associated file:

        - When the ADVANCING PAGE phrase of the WRITE statement is specified, the LINAGE-COUNTER is automatically reset to one. During the resetting of LINAGE-COUNTER to the value one, the value of LINAGE-COUNTER is implicitly incremented to exceed the value specified by integer-1 or the data item referenced by data-name-1.

        - When the ADVANCING identifier-2 or integer-1 phrase of the WRITE statement is specified, the LINAGE-COUNTER is incremented by integer-1 or the value of the data item referenced by identifier-2.

        - When the ADVANCING phrase of the WRITE statement is not specified, the LINAGE-COUNTER is incremented by the value one.

        - The value of LINAGE-COUNTER is automatically reset to one when the device is repositioned to the first line that may be written on for each of the succeeding logical pages.

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

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

11. If the file connector associated with this file description entry is an external file connector, all file description entries in the run unit that are associated with this file connector shall have:
    a. A LINAGE clause, if any file description entry has a LINAGE clause.
    b. The same corresponding values for integer-1, integer-2, integer-3, and integer-4, if specified.
    c. The same corresponding external data items referenced by data-name-1, data-name-2, data-name-3, and data-name-4.

LINE Clause

The LINE clause specifies vertical positioning information for its screen item.

General format

[AT|FROM] {LINE|ROW} NUMBER [PLUS|+] {identifier-1 | integer-1} [CELL|CELLS| PIXEL|PIXELS]

Syntax rules

1. PLUS and + are synonyms.
2. Identifier-1 shall be described in the file, working-storage, local-storage, or linkage section as an elementary unsigned integer data item.
3. The PLUS phrase shall not be specified for the first elementary item in a screen record.
4. CELL and CELLS are synonymous. It applies only to graphical screen items.
5. PIXEL and PIXELS are synonymous. It applies only to graphical screen items.

General rules

1. The LINE clause, in conjunction with the COLUMN clause, establishes the starting coordinates for a screen item within a screen record. The LINE clause specifies the vertical coordinate. Positioning of the screen record and within the screen record appears the same on the terminal display regardless of whether the whole screen record or just a portion of it is referenced in an ACCEPT screen or a DISPLAY screen statement.
2. If the LINE clause does not specify PLUS, the clause gives the line number relative to the start of the screen record. A line number of 1 represents the first line of the screen record.
3. If the PLUS phrase is specified in the LINE clause, the line number is relative to the end of the preceding screen item in the same screen record. PLUS denotes a line position that is increased by the value of identifier-1 or integer-1.
4. If the LINE clause is omitted, the following apply:
    a. if no previous screen item has been defined, LINE 1 of the screen record is assumed.
    b. if a previous screen item has been defined, the line of that previous item is assumed.

5. CELL|CELLS is redundant; the default positioning is based on cells.

5. PIXEL|PIXELS specifies the value in terms of pixels directly allowing directly control over placement. Avoid direct pixel control where possible.

LINES Clause Graphical

Specify the vertical size of a graphical component. The control units for liens are dependent upon the component, but are typically expressed in terms of the height of a character in the component's font. The lines may be expressed directly in terms of text lines or pixels.

General format

LINES [IS|=] height [CELL|CELLS|PIXEL|PIXELS]

Syntax rules

1. CELL and CELLS are synonymous.
2. PIXEL and PIXELS are synonymous.
3. Height must be numeric, but may be non-integer; graphical screen items may have non-integer heights.

LOWER Clause

The LOWER clause is syntax-checked only for textual screen items.

General format

LOWER

Syntax rules

For textual screens, the LOWER clause is ignored.

LOWLIGHT Clause

The LOWLIGHT clause specifies that the field is to appear on the screen with the lowest level of intensity.

General format

WITH {LOWLIGHT|DIM|LOW}

General rules

1. A LOWLIGHT clause specified at the group level applies to each elementary screen item in that group.
2. When the LOWLIGHT clause is specified, the characters that constitute the screen item will be displayed in the foreground color at the lowest intensity.

NUMERIC-FILL Clause

The NUMERIC-FILL clause is syntax-checked only for textual screen items.

General format

WITH NUMERIC-FILL

OCCURS Clause

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

General format

Format 1 (fixed-table):

OCCURS integer-2 TIMES
[{ASCENDING|DESCENDING} KEY IS {data-name-2}…] …
[INDEXED BY {index-name-1}…]

Format 2 (variable-table):

OCCURS [integer-1 TO] integer-2 TIMES DEPENDING ON data-name-1
[{ASCENDING|DESCENDING} KEY IS {data-name-2} … } …
[INDEXED BY {index-name-1}…]

Syntax rules

1. The OCCURS clause shall not be specified in a data description entry that:
   a. Has a level-number of 01, 66, 77, or 88, or
   b. Has a variable-occurrence data item subordinate to it.
2. If the DEPENDING ON phrase is not specified, an OCCURS clause may be subordinate to a data description entry that contains another OCCURS clause as long as the number of subscripts required does not exceed seven.
3. Data-name-1 and data-name-2 may be qualified.
4. The first specification of data-name-2 shall be the name of either the entry containing the OCCURS clause or an entry subordinate to the entry containing the OCCURS clause. Subsequent specification of data-name-2 shall be subordinate to the entry containing the OCCURS clause.
5. Data-name-2 shall be specified without the subscripting normally required.
6. In format 2, integer-1 shall be greater than or equal to zero and integer-2 shall be greater than integer-1.
7. Data-name-1 shall describe an integer.
8. The data item defined by data-name-1 shall not occupy a character position within the range of the first character position defined by the data description entry containing the OCCURS clause and the last character position defined by the record description entry containing that OCCURS clause.

9. If the OCCURS clause is specified in a data description entry included in a record description entry containing the EXTERNAL clause, data-name-1, if specified, shall reference a data item possessing the external attribute that is described in the same data division.
10. If the OCCURS clause is specified in a data description entry subordinate to one containing the GLOBAL clause, data-name-1, if specified, shall be a global name and shall reference a data item that is described in the same data division.
11. A data description entry that contains format 2 of the OCCURS clause may only be followed, within that record description, by data description entries that are subordinate to it.
12. The data item identified by data-name-2 shall not contain an OCCURS clause except when data-name-2 is the subject of the entry.
13. There shall not be any entry that contains an OCCURS clause between the descriptions of the data items identified by the data-names in the KEY IS phrase and the subject of the entry.
14. The INDEXED phrase shall be specified if the subject of this entry, or an entry subordinate to this entry, is to be referenced by subscripting using an index-name or if a SEARCH ALL statement references the subject of this entry as the subject table to be searched.
15. Index-name-1 shall be specified only in the following contexts:
    - as a subscript;
    - in the varying phrase of a PERFORM statement;
    - in the varying phrase of a search statement;
    - in the set statement;
    - as an operand in a relation condition.
16. Integer-4 shall be greater than integer-2 and greater than integer-3.
17. The KEY clause shall not be specified for a data item of class object or pointer.
18. Screen description entries should not includes the OCCURS clause.

General rules

ALL FORMATS

1. Except for the OCCURS clause itself, all data description clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described.
2. The number of occurrences of the subject entry is defined as follows:
    a. In format 1, the value of integer-2 represents the exact number of occurrences.
    b. In format 2, the value of the data item referenced by data-name-1 represents the current number of occurrences.

This format specifies that the subject of this entry has a variable number of occurrences. The value of integer-2 represents the maximum number of occurrences and the value of integer-1 represents the minimum

number of occurrences. This does not imply that the length of the subject of the entry is variable, but that the number of occurrences is variable.

At the time the subject of entry is referenced or any data item subordinate or superordinate to the subject of entry is referenced, the value of the data item referenced by data-name-1 shall fall within the bounds from integer-1 through integer-2. If the value of the data item does not fall within the specified bounds, the EC-BOUND-ODO exception condition exists. The content of a data item whose occurrence number exceeds the value of the data item referenced by data-name-1 are undefined.

FORMATS 1 AND 2

1. The allocation and format of the index defined by index-name-1 are dependent on the implementor and the hardware. The implementor shall specify the rules for the range of values allowed in the index defined by index-name-1. This range shall include 0 through and including the value that represents integer-2 plus 1. An index may be modified only by a PERFORM VARYING statement, a SEARCH statement, and a SET statement.

   If one of these statements causes the index to be set to a value outside of the values allowed by the implementor, the EC-RANGE-INDEX exception condition exists.

2. The KEY phrase indicates the order of key data items used during execution of a SEARCH statement with the ALL phrase specified, or during the execution of a SORT statement that references a table. If more than one data-name-2 is specified, they are specified in descending order of significance. The data associated with data-name-2 shall be ordered when a SEARCH statement with the ALL phrase is executed if data-name-2 is specified in one of the conditions in the WHEN phrase. At the time of the execution of such a SEARCH statement, the contents of the data items referenced by data-name-2 shall be in ascending order if the ASCENDING phrase is specified or descending order if the DESCENDING phrase is specified. The associated collating sequence for the order is determined by the rules for comparison of operands that apply to the condition specified in the WHEN phrase of the SEARCH statement.

3. During a DISPLAY screen or an ACCEPT screen statement that references a screen item whose description includes the OCCURS clause and whose description or whose subordinate's description includes a FROM, TO, or USING clause, the data values for corresponding table elements are moved from the data table element to the screen table element or from the screen table element to the data table element.

4. If the description of a screen item includes the OCCURS clause, the positioning within the screen record of each occurrence of that screen item is as follows:

   a. If the description of that screen item contains a COLUMN clause, each occurrence behaves as though it had the same COLUMN clause specified.

   b. If that screen item is a group item with a subordinate screen item whose description contains a COLUMN clause with the PLUS or MINUS phrase and

that group screen item is subordinate to a screen item whose description contains a LINE clause, each occurrence behaves as though it had the same subordinate entries with the same COLUMN clause specified.

c. If the description of that screen item contains a LINE clause with the PLUS or MINUS phrase, each occurrence behaves as though it had the same LINE clause specified.

d. If that screen item is a group item with a subordinate screen item whose description contains a LINE clause with the PLUS or MINUS phrase, each occurrence behaves as though it had the same subordinate entries with the same LINE clause specified.

FORMAT 2

1. When a group data item, having subordinate to it an entry that specifies format 2 of the OCCURS clause, is referenced, the part of the table area used in the operation is determined as follows:

   a. If the data item referenced by data-name-1 is outside the group, only that part of the table area that is specified by the value of the data item referenced by data-name-1 at the start of the operation will be used.

   b. If the data item referenced by data-name-1 is included in the same group and the group data item is referenced as a sending item, only that part of the table area that is specified by the value of the data item referenced by data-name-1 at the start of the operation will be used in the operation. If the group is a receiving item, the maximum length of the group will be used.

2. If format 2 is specified in a record description entry and the associated file description or sort-merge description entry contains the VARYING phrase of the RECORD clause, the records are variable length. If the DEPENDING ON phrase of the RECORD clause is not specified, the content of the data item referenced by data-name-1 of the OCCURS clause shall be set to the number of occurrences to be written before the execution of any RELEASE, REWRITE, or WRITE statement.

3. If integer-2 is specified, but not integer-1, then integer-1 is assumed to be one (1).

OUTPUT JUSTIFIED Clause

The OUTPUT JUSTIFIED clause is syntax-checked only for textual screen items.

General format

{OUTPUT JUSTIFIED {LEFT|RIGHT|CENTERED}}
| {OUTPUT JUSTIFIED {LEFT|RIGHT|CENTERED}}

Syntax rules

For textual screens, the OUTPUT clause is ignored.

OVERLINE Clause

The OVERLINE clause specifies that each character of the field is overlined, rendered with a line above the character, when it is displayed on the screen.

General format

WITH OVERLINE

General rules

1. If the OVERLINE clause is specified at group level, it applies to each elementary screen item in that group.
2. When the OVERLINE clause is specified, the screen item will be displayed so that the characters that constitute the screen item are overlined.

PICTURE Clause

The PICTURE clause describes the general characteristics, editing requirements, format validation profile of an elementary item.

General format

{PICTURE|PIC} IS character-string

Syntax rules

1. A PICTURE clause may be specified only at the elementary item level.
2. A character-string consists of certain allowable combinations of characters in the COBOL character set used as symbols. The allowable combinations determine the category of the elementary item.
3. The lowercase letters corresponding to the uppercase letters representing the PICTURE symbols A, B, N, G, P, S, V, X, Z, CR, and DB are equivalent to their uppercase representations in a PICTURE character-string. All other lowercase letters are not equivalent to their corresponding uppercase representations.
4. The maximum number of characters allowed in the character-string is 50.
5. PIC is an abbreviation for PICTURE.
6. The asterisk when used as the zero suppression symbol and the clause BLANK WHEN ZERO may not appear in the same entry.
7. Picture symbols 'A' and 'X' shall not be specified when usage national is specified for the subject of the entry.

General rules

1. The PICTURE clause defines the following categories of data:
    - alphabetic
    - alphanumeric
    - alphanumeric-edited

- national
- national-edited
- numeric
- numeric-edited

2. To define an item as alphabetic:
   a. Its PICTURE character-string shall contain only the symbol 'A'; and
   b. Its content, when represented in standard data format, shall be one or more alphabetic characters.
3. To define an item as alphanumeric:
   a. Its PICTURE character-string shall contain any combination of symbols from the set 'A', 'X', and '9' that includes at least one 'X' or any two different symbols from this set.
   b. Its content when represented in standard data format shall be one or more characters in the computer's character set.
4. To define an item as alphanumeric-edited:
   a. Its PICTURE character-string shall contain any combination of symbols from the set 'A', 'X', '9', 'B', '0', and '/' that includes at least one 'A' or 'X' and one of the characters 'B', '0', or '/'.
   b. Its content when represented in standard data format shall be two or more characters in the computer's character set.
5. To define an item as national, its PICTURE character-string shall contain only the symbol 'N' or ‘G’.
6. To define an item as national-edited, its PICTURE character-string shall contain a combination of the symbols 'N', ‘G’, 'B', '0', and '/' and shall include at least one 'N' or ‘G’ and one of the characters 'B', '0', or '/'.
7. To define an item as numeric:
   a. Its PICTURE character-string shall contain only the symbols '9', 'P', 'S', and 'V'. The number of digit positions that may be described by the PICTURE character-string shall range from 1 to 18 inclusive; and
   b. If unsigned, its content when represented in standard data format shall be one or more numeric characters; if signed, the item may also contain a '+', '-', or other representation of an operational sign (see SIGN clause).
8. To define an item as fixed-point numeric-edited:
   a. Its PICTURE character-string is restricted to certain combinations of the symbols 'B', '/', 'P', 'V', 'Z', '0', '9', ',', '.', '*', '+', '-', 'CR', 'DB', and the currency symbol. The allowable combinations are determined from the order of precedence of symbols and the editing rules; and
      } The number of digit positions that may be represented in the PICTURE character-string shall range from 1 to 18 inclusive; and

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

b. The content of each of the character positions shall be consistent with the corresponding PICTURE symbol.

9. When a usage national clause is specified for the subject of the entry, for symbols representing a character position, a national character position shall be represented. For symbols representing characters to be inserted, a national insertion character shall be represented.

10. The size of an elementary item, where size means the number of character positions occupied by the elementary item in standard data format, is determined by the number of allowable symbols that represent positions. An unsigned nonzero integer that is enclosed in parentheses following the symbols 'A', ',', 'X', '9', 'N', 'G', 'P', 'Z', '*', 'B', '/', '0', '1', '+', '-', or the currency symbol indicates the number of consecutive occurrences of the symbol. The following symbols may appear only once in a given PICTURE: 'S', 'V', '.', 'CR', 'DB'.

11. The functions of the symbols used to describe an elementary item are explanation.

Elementary Item Symbols

A	Each 'A' in the character-string represents a character position that shall contain any character from the computer's alphanumeric character set and is counted in the size of the item.
B	Each 'B' in the character-string represents a character position into which the space character will be inserted and is counted in the size of the item.
N or G	Each 'N' or 'G' in the character-string represents a national character position that shall contain a character from the computer's national character set. Each 'N' or 'G' shall be counted in the size of the data item being described.
P	Each 'P' in the character-string indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position symbol 'P' is not counted in the size of the data item. Scaling position symbols are counted in determining the maximum number of digit positions (31) in numeric-edited items or numeric items. The scaling position symbol 'P' may appear only as a continuous string of 'P's in the leftmost or rightmost digit positions within a PICTURE character-string; since the scaling position symbol 'P' implies an assumed decimal point (to the left of 'P's if 'P's are leftmost PICTURE symbols and to the right if 'P's are rightmost PICTURE symbols), the assumed decimal point symbol 'V' is redundant as either the leftmost or rightmost symbols within such a PICTURE description. The symbol 'P' and the insertion symbol '.' (period) may not both occur in the same PICTURE character-string. In certain operations that reference a data item whose PICTURE character-string contains the symbol 'P', the algebraic value of the data item is used rather than the actual 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. An elementary 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	The 'S' is used in a character-string to indicate the presence, but neither the representation nor, necessarily, the position of an operational sign; it shall be written as the leftmost character in the PICTURE. The 'S' is not counted in determining the size (in terms of standard data format characters) of the elementary item unless the entry is subject to a SIGN clause that specifies the optional SEPARATE CHARACTER phrase.
V	The 'V' is used in a character-string to indicate the location of the assumed decimal point and may only appear once in a character-string. The 'V' does not represent a character position and therefore is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string representing a digit position or scaling position, the 'V' is redundant.
X	Each 'X' in the character-string represents an alphanumeric character position that shall contain any character from the computer's alphanumeric character set and shall be counted in the size of the data item being described. The characters represented may be graphic or non-graphic characters. If national characters in external media format are present in the storage referenced by an item defined as alphanumeric, the contents shall be treated as a string of alphanumeric characters except for the purposes of explicit conversion between classes as defined for the DISPLAY-OF and NATIONAL-OF functions.
Z	Each 'Z' in a character-string represents the leftmost leading numeric character position that will be replaced by a space character when the content of that character position is a leading zero. Each 'Z' is counted in the size of the item.
9	Each '9' in the character-string represents a digit position that contains a numeric character and is counted in the size of the item.
0	Each '0' (zero) in the character-string represents a character position into which the character zero will be inserted. The '0' is counted in the size of the item.
/	Each '/' (slant) in the character-string represents a character position into which the slant character will be inserted. The '/' is counted in the size of the item.
,	Each ',' (comma) in the character-string represents a character position into which the character ',' will be inserted. This character position is counted in the size of the item.
.	When the symbol '.' (period) appears in the character-string it is an editing symbol that represents the decimal point for alignment purposes and, in addition, represents a character position into which the character '.' will be inserted. The character '.' is counted in the size of the item. For a given program the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange the rules for the period apply to the comma and the rules for the comma apply to the period wherever they appear in a PICTURE clause.
+ - CR DB	These symbols are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol is placed. Each character used in the symbol is counted in determining the size of the item being described. In a floating-point PICTURE character-string, both a '+' and 'S' may appear; otherwise, these four symbols are mutually exclusive in any one PICTURE character-string.
*	Each '*' (asterisk) in the character-string represents a leading numeric character position into which an asterisk will be placed when the content of that position is a leading zero. Each '*' is counted in the size of the item.
cs	The currency symbol in the character-string represents character positions into which a currency string is to be placed. A currency symbol is represented in a PICTURE character-string by either the currency sign or by a currency symbol specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. The first occurrence of the currency symbol adds the number of characters in the currency string to the size of the item. Subsequent occurrences add one character to the size of the item.

12. The value of the insertion and replacement characters shall be the value of those characters in the COBOL character set. When the usage of the item being

described is national, the value used shall be the national character set representation of the insertion and replacement COBOL characters.

Editing rules

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

    There are two types of suppression and replacement editing:
    - Zero suppression and replacement with spaces
    - Zero suppression and replacement with asterisks
2. The type of editing that may be performed upon an item is dependent upon the category to which the item belongs. Table 6, Category and type of editing, specifies which type of editing may be performed upon a given category:

Category and type of editing

Category	Type of editing
Alphabetic	None
Alphanumeric	None
National	None
Numeric	None
Alphanumeric-edited	Simple insertion '0', 'B', and '/'
National-edited	Simple insertion '0', 'B', and '/'
Numeric-edited	All, subject to rules in editing rule 3 below

3. Floating insertion editing and editing by zero suppression and replacement are mutually exclusive in a PICTURE clause. Only one type of replacement may be used with zero suppression in a PICTURE clause.
4. Simple insertion editing. The ',' (comma), 'B' (space), '0' (zero), and '/' (slant) are used as the insertion symbols. The insertion symbols are counted in the size of the item and represent the position in the item into which the associated character will be inserted. If the insertion symbol ',' (comma) is the last symbol in the PICTURE character-string, the PICTURE clause shall be the last clause of the data description entry and shall be immediately followed by the separator period. This results in the combination of ',.' appearing in the data description entry, or, if the DECIMAL-POINT IS COMMA clause is used, in two consecutive periods.
5. Special insertion editing. The '.' (period) is used as the insertion symbol. It also represents the decimal point for alignment purposes. The insertion character used for the actual decimal point is counted in the size of the item. The use of the assumed decimal point, represented by the symbol 'V' and the actual decimal point, represented by the insertion character, in the same PICTURE character-string is disallowed. If the insertion character is the last symbol in the PICTURE character-string, the PICTURE clause shall be the last

clause of that data description entry and shall be immediately followed by the separator period. This results in two consecutive periods appearing in the data description entry, or in the combination of ',.' if the DECIMAL-POINT IS COMMA clause is used. The result of special insertion editing is the appearance of the insertion character in the item in the same position as specified in the character-string.

6. Fixed insertion editing. The currency symbol and the editing sign control symbols '+', '-', 'CR', 'DB' are the insertion symbols. Only one currency symbol and only one of the editing sign control symbols may be used in a given PICTURE character-string. When the symbols 'CR' or 'DB' are used they represent two character positions in determining the size of the item and they represent the rightmost character positions that are counted in the size of the item. The uppercase letters 'CR' or 'DB' are the insertion characters. The symbol '+' or '- ', when used, shall be either the leftmost or rightmost character position to be counted in the size of the item. The currency symbol shall be either the leftmost or the rightmost character, with the exception of an optional '+' or '-' symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as the associated symbol occupies in the PICTURE character-string. Editing sign control symbols produce the following results depending upon the value of the data item:

Results of fixed insertion editing

Editing symbol in PICTURE character-string	Result	
	Data item positive or zero	Data item negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

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

   Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two of the floating insertion symbols. Any of the simple insertion symbols embedded in the string of symbols or to the immediate right of this string are part of the string. When the floating insertion symbol is the currency symbol, this string of floating insertion symbols may have one of the editing sign control symbols '+, '-', 'CR', or 'DB' immediately to the right of this string.

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

   The second floating symbol from the left represents the leftmost limit of the numeric data that may be stored in the data item. Nonzero numeric data may replace all the symbols at or to the right of this limit.

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

If the insertion character positions are only to the left of the decimal point in the PICTURE character-string, the result is that a single occurrence of the replacement characters is placed into the character position or positions immediately preceding either the decimal point or the first non-zero digit in the data, whichever is farther to the left in the PICTURE character-string. Any character positions preceding these insertion characters are replaced with spaces.

If all numeric symbol positions in the PICTURE character-string are represented by the insertion symbol, at least one of the insertion symbols shall be to the left of the decimal point.

When the floating insertion symbol is the editing control symbol '+' or '-', the character inserted depends upon the value of the data item:

Results of floating insertion editing

Editing symbol in PICTURE character-string	Result	
	Data item positive or zero	Data item negative
+	+	-
-	space	-

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

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item shall be the number of characters in the sending data item, plus the number of non floating insertion symbols being edited into the receiving data item, plus one for the floating insertion symbol. If truncation does occur, the value of the data that is used for editing is the value after truncation. (See 13.2.8, Standard alignment rules.)

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

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

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

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

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

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

Precedence rules

Table 9, PICTURE symbol order of precedence, shows the order of precedence of symbols in a character-string. An 'X' at an intersection indicates that the symbol(s) at the top of the column may precede (but not necessarily immediately), in a given character-string, the symbol(s) at the left of the row. The currency symbol is indicated by the symbol 'cs'.

At least one of the symbols 'A', 'N', 'X', 'Z', '1', '9', or '*' or at least two occurrences of one of the symbols '+', '-', or 'cs' shall be present in a PICTURE character-string.

Nonfloating insertion symbols '+', floating insertion symbols 'Z', '*', '+', '-', and 'cs', and other symbol 'P' appear twice in table 9, PICTURE symbol order of precedence. 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 chart represents its use to the right of the decimal point position.

PICTURE symbol order of precedence

First Symbol		Non-floating Insert symbols										Floating Insertion symbols						Other Symbols							
Second Symbol		B	O	/	,	.	+ -	+ -	CR DB	C S	C S	Z *	Z *	+ -	+ -	C S	C S	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
	O	X	X	X	X	X	X			X		X	X	X	X	X	X	X	X		X		X		X
	/	X	X	X	X	X	X			X		X	X	X	X	X	X	X	X		X		X		X
	,	X	X	X	X	X	X			X		X	X	X	X	X	X	X			X		X		X
	.	X	X	X	X		X			X		X		X		X		X							
	+ -																								
	+ -	X	X	X	X	X				X		X	X			X	X	X			X	X	X		
	CR DB	X	X	X	X	X				X		X	X			X	X	X			X	X	X		
	cs						X																		
	cs	X	X	X	X	X	X	X				X	X			X	X	X			X	X	X		
Floating Insertion Symbols	Z *	X	X	X	X	X		X		X		X													
	Z *	X	X	X	X	X	X	X		X		X	X								X		X		
	+ -	X	X	X	X					X				X											
	+ -	X	X	X	X	X				X				X	X						X				
	cs	X	X	X	X		X									X									
	cs	X	X	X	X	X	X									X	X				X				
Other Symbols	9	X	X	X	X	X	X			X		X		X		X	X	X	X		X		X		
	A 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																							X	
	GN	X	X	X																					X

Note - When two picture symbols appear together in the same row or column, the symbols are mutually exclusive.

PROCEDURE Clause

The procedure clause links a set of paragraphs or sections to the before, after, exception or event handler for a graphical component.

General format

BEFORE PROCEDURE {{entry-1 [THROUGH entry-2]} | NULLS | NULL}
AFTER PROCEDURE {{entry-1 [THROUGH entry-2]} | NULLS | NULL}
EXCEPTION PROCEDURE {{entry-1 [THROUGH entry-2]} | NULLS | NULL}
EVENT PROCEDURE {{entry-1 [THROUGH entry-2]} | NULLS | NULL}

Syntax rules

1. NULL or NULLS is commentary, as the default is NULL; this means that there is no procedure to be activated.
2. If entry-2 is specified, entry-1 must precede entry-2 in the program.
3. For textual screens, the PROCEDURE clause is ignored.

PROMPT Clause

The PROMPT clause determines the character used to fill empty space positions in an active input field.

General format

WITH [NO] PROMPT CHARACTER [identifier-1|literal-1]

General rules

1. The PROMPT clause only has an effect during the execution of an ACCEPT statement referencing the screen item.
2. The PROMPT clause only has an effect on the active input item.
3. NO PROMPT specifies that the space character is the prompt character.
4. If neither identifier-1 nor literal-1 is specified, the default is used. The default character is currently the underscore (_) character.
5. PROMPT characters are set to spaces at termination of the active input field.

Property Name Clause

Specify a property name and value to a graphical component. Many property-names are shared among all handle-components, while some are specific to an individual handle-component. See the Appendix for the complete list of recognized property names and values.

General format

[property-name [IS|=] {property-value | (property-value…) | {MULTIPLE|TABLE} property-table}]

[property-name]

[{NOT|NO} property-name]

[PROPERTY integer [IS|=] {property-value | (property-value…) | {MULTIPLE|TABLE} property-table }]

Syntax rules

1. property-name must be a valid property-name for the current handle-component. That is, it must be in the list in the Appendix.
2. If multiple property values are specified in parenthesis, then each value is set in turn. Some property names take multiple values, allowing an easier set mechanism.
3. If MULTIPLE or TABLE is specified, then the items are retrieved from the given array name.
4. If a property-name is a Boolean property-name, then just specifying the name is sufficient to set it to true; specifying NOT property-name or NO property-name will set the property to false. A Boolean property-name is also known as a style.

General rules

1. The property of the graphical screen item currently being defined is set to the given property value.
2. The meaning of this set is the same as the meaning given by the MODIFY verb.
3. A property name will generally be an indication of the intended function of the property.
4. The setting takes place upon a DISPLAY, not during initialization.
5. The setting is reapplied for each DISPLAY; some properties are cumulative and should be reset before a subsequent DISPLAY.

RECORD Clause

The RECORD clause specifies the number of alphanumeric character positions in a fixed length record, or specifies the range of alphanumeric character positions in a variable-length record. If the number of alphanumeric character positions does vary, the clause specifies the minimum and maximum number of alphanumeric character positions.

General format

Format 1 (fixed-length):

RECORD {CONTAINS integer-1 CHARACTERS}

Format 2 (variable-length):

RECORD IS VARYING IN SIZE [[FROM integer-2] [TO integer-3] CHARACTERS] [DEPENDING ON data-name-1]

Format 3 (fixed-or-variable-length):

RECORD CONTAINS integer-4 TO integer-5 CHARACTERS

Syntax rules

ALL FORMATS

If no record description entries are specified in a file description entry for a file other than a report file, the RECORD clause shall be specified.

FORMAT 1

No record description entry for the file may specify a number of alphanumeric character positions greater than integer-1.

FORMAT 2

1. Record descriptions for the file shall not describe records that contain a lesser number of alphanumeric character positions than that specified by integer-2 nor records that contain a greater number of alphanumeric character positions than that specified by integer-3.
2. Integer-3 shall be greater than integer-2.
3. Data-name-1 shall describe an elementary unsigned integer in the working-storage, local-storage, or linkage section.

General rules

ALL FORMATS

1. Each integer in a RECORD clause specifies a record size in terms of alphanumeric character positions.
2. The implicit or explicit RECORD clause specifies the size of records in the record area. The size of records on physical storage media may be different due to control information required by the operating environment. Factors in the program other than the RECORD clause that may affect the size of records on physical storage medium are the CODE-SET clause. The size on the physical medium varies according to the organization of the file and the virtual file system used which may vary at runtime.
3. The size of each data record is specified in terms of the number of alphanumeric character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of alphanumeric character positions in all fixed length elementary items plus the sum of the maximum number of alphanumeric character positions in any variable-length item subordinate to the record. Implicit filler positions, if any, are included in the size.
4. If the RECORD clause is not specified, an implicit format 1 or format 2 RECORD clause is assumed to be specified. This implicit RECORD clause is defined by the implementor with the following characteristics:
    a. If format 1 is implied, integer-1 shall be the record size of the largest record description entry in this file description entry.
    b. If format 2 is implied, integer-2 shall be the record size of the smallest record description entry in this file description entry, and integer-3 shall be the largest record description entry in this file description entry. The DEPENDING ON phrase is assumed to be omitted.
5. 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 shall specify the

same values for integer-1 or integer-2 and integer-3. If the RECORD clause is not specified, all record description entries associated with this file connector shall be the same length.

FORMAT 1

Format 1 is used to specify fixed-length records. Integer-1 specifies the number of alphanumeric character positions contained in each record in the file.

FORMAT 2

1. Format 2 is used to specify variable-length records. Integer-2 specifies the minimum number of alphanumeric character positions to be contained in any record of the file. Integer-3 specifies the maximum number of alphanumeric character positions in any record of the file.
2. The number of alphanumeric character positions associated with a record description is determined by the sum of the number of alphanumeric character positions in all elementary data items excluding redefinition and renaming, plus any implicit FILLER due to synchronization. If a table is specified:
    a. The maximum number of table elements described in the record is used in the summation above to determine the maximum number of alphanumeric character positions associated with the record description.
3. If integer-2 is not specified, the minimum number of alphanumeric character positions to be contained in any record of the file is equal to the least number of alphanumeric character positions described for a record in that file.
4. If integer-3 is not specified, the maximum number of alphanumeric character positions to be contained in any record of the file is equal to the greatest number of alphanumeric character positions described for a record in that file.
5. If data-name-1 is specified, the number of alphanumeric character positions in the record shall be placed into the data item referenced by data-name-1 before any RELEASE, REWRITE, or WRITE statement is executed for the file.
6. If data-name-1 is specified, the execution of a DELETE, RELEASE, REWRITE, START, or WRITE statement or the unsuccessful execution of a READ or RETURN statement does not alter the content of the data item referenced by data-name-1.
7. During the execution of a RELEASE, REWRITE, or WRITE statement, the number of alphanumeric character positions in the record is determined by the following conditions:
    a. If data-name-1 is specified, by the content of the data item referenced by data-name-1.
    b. If data-name-1 is not specified and the record does not contain a variable-occurrence data item, by the number of alphanumeric character positions in the record.
    c. If data-name-1 is not specified and the record does contain 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.

8. If the number of alphanumeric character positions in the logical record to be written is less than integer-2 or greater than integer-3, the following occurs:
    a. If a REWRITE or WRITE statement is being executed, the execution of the REWRITE or WRITE statement is unsuccessful.
    b. If a RELEASE statement is being executed, the execution of the RELEASE statement is unsuccessful.
9. If data-name-1 is specified, after the successful execution of a READ or RETURN statement for the file, the contents of the data item referenced by data-name-1 will indicate the number of alphanumeric character positions in the record just read.
10. If the INTO phrase is specified in the READ or RETURN statement, the number of alphanumeric character positions in the current record that participate as the sending data items in the implicit MOVE statement is determined by the following conditions:
    a. If data-name-1 is specified, by the content of the data item referenced by data-name-1.
    b. If data-name-1 is not specified, by the value that would have been moved into the data item referenced by data-name-1 had data-name-1 been specified.

FORMAT 3

1. The virtual file system in use by the runtime defines whether format 3 of the RECORD clause produces fixed length records or variable-length records.
   The Elastic COBOL file system uses variable-length if integer-4 and integer-5 are different from one another; fixed-length is used if integer-4 is equal to integer-5.
   The Micro Focus compatible file system uses fixed length.
2. When format 3 of the RECORD clause is used, integer-4 and integer-5 refer to the minimum number of alphanumeric characters in the smallest size data record and the maximum number of alphanumeric characters in the largest size data record, respectively. However, in this case, the size of each data record is completely defined in the record description entry.
3. If the number of alphanumeric character positions in the logical record to be written is less than integer-4 or greater than integer-5, the following occurs:
    a. If a REWRITE or WRITE statement is being executed, the execution of the REWRITE or WRITE statement is unsuccessful.
    b. If a RELEASE statement is being executed, the execution of the RELEASE statement is unsuccessful.

REDEFINES Clause

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

General format

level-number [data-name-1|FILLER] REDEFINES data-name-2

NOTE - Level-number, data-name-1, and FILLER are shown in the above format to provide context. Level-number, data-name-1, and FILLER are not part of the REDEFINES clause.

Syntax rules

1. The REDEFINES clause, when specified, shall immediately follow the subject of the entry.
2. The level-numbers of data-name-2 and the subject of the entry shall be identical, but shall not be 66 or 88.
3. This clause shall not be specified in level 1 entries in the file section or in any entry in a file description entry that contains a FORMAT clause.
4. This clause shall not be specified in level 1 entries in the communication section.
5. No entry having a level-number numerically lower than the level-number of data-name-2 may occur between the data description entries of data-name-2 and the subject of the entry.
6. The data description entry for data-name-2 shall not contain an OCCURS clause. However, data-name-2 may be subordinate to an item whose data description entry contains an OCCURS clause. In this case, the reference to data-name-2 in the REDEFINES clause may not be subscripted. Neither the original definition nor the redefinition may include a variable-occurrence data item.
7. Data-name-2 shall not be qualified even if it is not unique since no ambiguity of reference exists in this case because of the required placement of the REDEFINES clause within the source program.
8. Multiple redefinition of the same storage area shall each specify as data-name-2 the data-name of the entry that originally defined the area.
9. The entries giving the new description of the storage area shall not contain any VALUE clauses, except in condition-name entries.
10. The entries giving the new descriptions of the storage area shall follow the entries defining the area of data-name-2, without intervening entries that define new storage areas.
11. Data-name-2 may be subordinate to an entry that contains a REDEFINES clause.
12. The REDEFINES clause shall not be specified for a data item of class object or pointer.
13. Data-name-2 shall not be of class object or pointer.

General rules

1. Storage allocation for the subject of the entry starts at data-name-2 and continues over a storage area sufficient to contain the number of bits required by the data item referenced by the subject of the entry. If the subject of the entry requires more bits than data-name-2, the storage area allocated for data-name-2 and the subject of the entry is the number of bits required by the data item referenced by the subject of the entry. The size used for references to data-name-2 is not changed.
2. When the same storage area is defined by more than one data description entry, the data-name associated with any of those data description entries may be used to reference that storage area.

RENAMES Clause

The RENAMES clause permits alternative, possibly overlapping, groupings of elementary items.

General format

66 data-name-1 RENAMES data-name-2 [{THROUGH|THRU} data-name-3].

NOTE - Level-number 66 and data-name-1 are shown in the above format to provide context. Level-number and data-name-1 are not part of the RENAMES clause.

Syntax rules

1. Any number of RENAMES entries may be written for a logical record.
2. All RENAMES entries referring to data items within a given logical record shall immediately follow the last data description entry of the associated record description entry.
3. Data-name-1 may not be used as a qualifier, and may only be qualified by the names of the associated level 01, FD, or SD entries. Neither data-name-2 nor data-name-3 may have an OCCURS clause in its data description entry nor be subordinate to an item that has an OCCURS clause in its data description entry.
4. Data-name-2 and data-name-3 shall be names of elementary items or groups of elementary items in the same logical record, and may not be the same data-name.
5. A 66 level entry may not rename another 66 level entry nor may it rename a 77, 88, or 01 level entry.
6. Data-name-2 and data-name-3 may be qualified.
7. None of the items within the range, including data-name-2 and data-name-3, if specified, shall be of class object or pointer, a strongly typed item, an item subordinate to a strongly typed item, or a variable-occurrence data item.
8. The words THROUGH and THRU are equivalent.
9. The beginning of the area described by data-name-3 shall not be to the left of the beginning of the area described by data-name-2. The end of the area described by data-name-3 shall be to the right of the end of the area described

by data-name-2. Data-name-3, therefore, shall not be subordinate to data-name-2.

General rules

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

REQUIRED Clause

The REQUIRED clause specifies that in the context of an ACCEPT screen statement, the user shall enter at least one character in the input field.

General format

WITH {REQUIRED|EMPTY-CHECK}

General rules

1. The REQUIRED clause only has an effect during the execution of an ACCEPT statement referencing the screen item.
2. The REQUIRED clause has no effect until the cursor enters a screen item subject to the REQUIRED clause.
3. The effect of the REQUIRED clause is to reject the terminator keystroke and any other cursor-moving keystrokes that would cause the cursor to move to another screen item unless the required termination condition is satisfied. The required termination condition for an alphanumeric screen item is that the screen item contains at least one non-space character. The required termination condition for a numeric or numeric-edited screen item is that the screen item contains a non-zero value.
4. For fields that are both input and output, the REQUIRED clause may be satisfied by the contents of the identifier or literal referenced in the FROM or USING clause, as well as data keyed by the terminal operator.
5. The REQUIRED clause is not effective if a function key is used to terminate the execution of the ACCEPT statement.
6. The specification of the FULL and REQUIRED clauses together requires that the field shall always be entirely filled before the normal termination key has any effect.
7. If a REQUIRED clause is specified in a group screen item, it applies to each elementary input screen item in that group.

REVERSE-VIDEO Clause

The REVERSE-VIDEO clause specifies that the screen item is to be displayed by exchanging the foreground and background colors that would otherwise be in effect.

General format

WITH {REVERSE-VIDEO|REVERSE|REVERSED}

General rules

1. If the REVERSE-VIDEO clause is specified at group level, it applies to each elementary screen item in that group.
2. When the REVERSE-VIDEO clause is specified, the screen item will be displayed so that the characters that constitute the screen item will be shown with the foreground and background colors being exchanged. For monochrome displays, the reverse-video attribute will be used.

AME Clause

The SAME clause is syntax-checked only for textual screen items.

General format

SAME

Syntax rules

For textual screens, the SAME clause is ignored.

SCROLL Clause

The SCROLL clause is syntax-checked only for textual screen items.

General format

SCROLL {UP|DOWN} BY integer-2 {LINE|LINES}

Syntax rules

For textual screens, the SCROLL clause is ignored.

SECURE Clause

The SECURE clause prevents data entered from the keyboard or contained in the screen item from appearing on the screen at the screen location that corresponds to the screen item for which it is specified.

General format

WITH {SECURE|NO-ECHO}

General rules

1. If a SECURE clause is specified in a group screen item, it applies to each elementary input screen item in that group.
2. The SECURE clause has an effect only during the execution of an ACCEPT statement referencing the screen item.
3. The effect of the SECURE clause is to prevent data corresponding to a screen item that has been specified with the SECURE clause from being displayed on the screen. During the execution of an ACCEPT statement, the cursor will appear at the screen location that corresponds to an input screen item, but any data keyed by the terminal operator will not be displayed. For fields that are both input and output, the contents of the screen location of the screen item prior to the execution of the ACCEPT screen statement remain unchanged and are unchangeable by the terminal operator.
4. Elastic COBOL displays an asterisk (*) character for each character typed in a secure field to track amount of data entered and cursor position.

SET-PROPERTY Clause

The SGET-PROPERTY clause generates two set methods for the identifier, allowing its value to be set as a Java Bean property by an external Elastic COBOL or Java program. This clause is suitable for enhanced Elastic COBOL/Java integration.

General format

SET-PROPERTY

Syntax rules

1. Only one SET-PROPERY may be specified per identifier.
2. The SET-PROPERTY may be specified in a PROGRAM-ID style program.
3. A single identifier may have both a GET-PROPERTY and a SET-PROPERTY.

General rules

1. Two set methods are generated for each identifier with a SET-PROPERTY.
2. If the COBOL variable must be subscripted for access, then integer subscripts, one per dimension of the table, are generated in the Java method signature, before the type; all indexes are based at one.
3. The first method is setName(String) takes a String, as if suitable for a MOVE for most types or Objects for object references, the second is setName(type) which returns a Java type natural to the COBOL type. The first letter of Name is capitalized, as is any letter after a '-'; any '-' is removed from the name. The type is given by the following table.

COBOL Type	setName(String)	setName([int dimension[,dimension]…]type)
Display	String	java.math.BigDecimal
Comp-D	String	java.math.BigDecimal
Packed-Decimal	String	java.math.BigDecimal
Binary	String	byte (1 byte), short (2 bytes), int (3-4 bytes), long

COBOL Type	setName(String)	setName([int dimension[,dimension]…]type)
		(5-8 bytes)
Binary-Rev	String	byte (1 byte), short (2 bytes), int (3-4 bytes), long (5-8 bytes)
Comp-X	String	byte (1 byte), short (2 bytes), int (3-4 bytes), long (5-8 bytes)
Comp-X-Rev	String	byte (1 byte), short (2 bytes), int (3-4 bytes), long (5-8 bytes)
Comp-S	String	short
Comp-1	String	float
Comp-1-MVS	String	float
Comp-1-REV	String	float
Comp-2	String	double
Comp-2-MVS	String	double
Comp-2-REV	String	double
Numeric-Edited	String	String
Alphanumeric	String	String
Alphanumeric-Edited	String	String
Alphabetic	String	String
National	String	String
National-Edited	String	String
Index	int	int
Object Reference	Object Class	Object Class
Numeric-Hashtable	Hashtable	java.util.Hashtable
Alphanumeric-Hashtable	Hashtable	java.util.Hashtable
Object-Hashtable	Hashtable	java.util.Hashtable
Jbyte	byte	byte
Jshort	short	short
Jint	int	int
Jlong	long	long
Jfloat	float	float
Jdouble	double	double
Jboolean	Boolean	Boolean
Handle	Object	Object

Example:

01 alpha pic 999 set-property.

01 beta pic 999 binary value 0 set-property.

01 gamma pic 999 comp-1 set-property.

01 delta pic 999 comp-2 set-property.

01 my-string pic x(10) set-property.

01 my-object object reference "java.util.Vector" set-property.

Generates methods:

public void setAlpha(String paramValue)

public void setAlpha(java.math.BigDecimal paramValue)

public void setBeta(String paramValue)

public void setBeta(short paramValue)

public void setGamma(String paramValue)

public void setGamma(float paramValue)

public void setDelta(String paramValue)

public void setDelta(double paramValue)

public void setMyString(String paramValue)

public void setMyObject(java.util.Vector paramValue)

SIGN Clause

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

General format

SIGN IS [LEADING|TRAILING] [SEPARATE CHARACTER]

Syntax rules

1. The SIGN clause may be specified only for:
    - a numeric data description entry whose PICTURE contains the symbol 'S'
    - a numeric screen description entry whose PICTURE contains the symbol 'S'
    - a group item containing at least one such numeric data or screen description entry.
2. The numeric data description entries for which the SIGN clause is specified shall be implicitly or explicitly as usage display or national.
3. If the CODE-SET clause is specified in a file description entry, any signed numeric data description entries associated with that file description entry shall be described with the SIGN IS SEPARATE clause.

General rules

1. The SIGN clause specifies the position and the mode of representation of the operational sign for the numeric data, report group, or screen description entry to which it applies, or for each numeric data or screen description entry subordinate to the group to which it applies. The SIGN clause applies only to numeric data or screen description entries whose PICTURE contains the symbol 'S'; the 'S' indicates the presence of, but neither the representation nor the position of, the operational sign.
2. If a SIGN clause is specified in a group item subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in the subordinate group item takes precedence for that subordinate group item.
3. If a SIGN clause is specified in an elementary numeric data or screen description entry subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in the subordinate elementary numeric data or screen description entry takes precedence for that elementary numeric data or screen item.

4. A numeric data description entry whose PICTURE contains the symbol 'S', but to which no optional SIGN clause applies, has an operational sign, but neither the representation nor the position of the operational sign is specified by the symbol 'S'. By default, TRAILING is assumed, but this may be modified by the NUMERIC SIGN clause in the SPECIAL-NAMES section. General rules 5 through 7 do not apply to such signed numeric data items.
5. If the SEPARATE CHARACTER phrase is not specified, then:
    a. The operational sign is presumed to be associated with the leading (or, respectively, trailing) digit position of the data item to which it applies.
    b. The symbol 'S' in a PICTURE character-string is not counted in determining the size of the item in terms of standard data format characters.
6. If the SEPARATE CHARACTER phrase is present, then:
    a. The operational sign is presumed to be the leading (or, respectively, trailing) character position of the data item to which it applies; this character position is not a digit position.
    b. The symbol 'S' in a PICTURE character-string is counted in determining the size of the item in terms of standard data format characters.
    c. The operational signs for positive and negative are the standard data format characters '+' and '-', respectively.
7. Every numeric data description entry whose PICTURE contains the symbol 'S' is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

SIZE Clause

The SIZE clause specifies the visibly active size of its data item.

General format

WITH SIZE {integer-1|identifier-1}

General rules

1. The applicable size is integer-1 or the contents of identifer-1.
2. The SIZE clause is applicable only to elementary items.

SIZE Clause Graphical

Specify the horizontal size of a graphical component. The control units for size are dependent upon the component, but are typically expressed in terms of the width of a character in the component's font. The size may be expressed directly in terms of text columns or pixels.

General format

SIZE [IS|=] length [CELL|CELLS|PIXEL|PIXELS]

Syntax rules

1. CELL and CELLS are synonymous.
2. PIXEL and PIXELS are synonymous.
3. Length must be numeric, but may be non-integer; graphical screen items may have non-integer sizes.

SPACE-FILL Clause

The SPACE-FILL clause causes data in free-format non-edited numeric data items to appear on the screen with zero-suppression in all integer character positions.

General format

WITH SPACE-FILL

General rules

1. This clause is applicable only to numeric non-edited data items.
2. This clause is effective when initial data is displayed and when an ACCEPT into the data item is terminated.
3. This clause is treated as commentary.

SPECIAL-NAMES Clause

The SPECIAL-NAMES clause allows certain screen related identifiers to be specified directly in the data division rather than the SPECIAL-NAMES paragraph.

General format

IS SPECIAL-NAMES CRT STATUS

IS SPECIAL-NAMES SCREEN CONTROL

IS SPECIAL-NAMES EVENT STATUS

IS SPECIAL-NAMES CURSOR

Syntax rules

The meaning of each SPECIAL-NAMES clause is identical to the meaning in the SPECIAL-NAMES paragraph, except the specified identifier is the subject of the data definition.

General rules

Only one definition for each SPECIAL-NAMES type may be given, in either the SPECIAL-NAMES paragraph or inline in the data division.

STANDARD Clause

The STANDARD clause is syntax-checked only for textual screen items.

General format

STANDARD

STYLE Clause

Set the style property of a graphical component. The style property is a bit vector of Boolean properties. The various style numbers are listed in the appendix. Avoid this property and use the property styles directly if possible.

General format

STYLE [IS|=] style

SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the alignment of an elementary item on the natural boundaries of the computer memory (see Item alignment for increased object-code efficiency).

General format

{SYNCHRONIZED|SYNC} [LEFT|RIGHT]

Syntax rules

1. The SYNCHRONIZED clause shall be specified only for an elementary item.
2. SYNC is an abbreviation for SYNCHRONIZED.

General rules

1. This clause specifies that the subject data item is to be aligned in the computer such that no other data item occupies any of the character positions between the leftmost and rightmost natural boundaries delimiting this data item. If the number of character positions required to store this data item is less than the number of character positions between those natural boundaries, the unused character positions or portions thereof shall not be used for any other data item. The unused character positions are not included in the character positions redefined when the elementary item is the object of a REDEFINES clause. Such unused character positions, however, are included in:
   a. The size of any group item that contains the subject of the entry; and
   b. The number of character positions allocated when any such group item is the object of a REDEFINES clause.
2. SYNCHRONIZED not followed by either RIGHT or LEFT specifies that the elementary item is to be positioned between natural boundaries in such a way as to effect efficient utilization of the elementary data item.

3. SYNCHRONIZED LEFT specifies that the elementary item is to be positioned such that it will begin at the left character position of the natural boundary in which the elementary item is placed.
4. SYNCHRONIZED RIGHT specifies that the elementary item is to be positioned such that it will terminate on the right character position of the natural boundary in which the elementary item is placed.
5. Whenever a SYNCHRONIZED item is referenced in the source program, the original size of the item, as shown in the PICTURE clause, the USAGE clause, and the SIGN clause, is used in determining any action that depends on size.
6. If the data description of an item contains an operational sign and any form of the SYNCHRONIZED clause, the sign of the item appears in the sign position explicitly or implicitly specified by the SIGN clause.
7. When the SYNCHRONIZED clause is specified in the data description entry that contains an OCCURS clause, or in the data description entry of a data item subordinate to the data description entry that contains an OCCURS clause, then:
   a. The SYNCHRONIZED clause applies to each occurrence of the data item.
   b. Any implicit FILLER generated for other data items within that same table is generated for each occurrence of those data items (see general rule 8b).
8. SYNCHRONIZED is treated as commentary.

SYSTEM MENU Clause

Set the system menu property of a graphical component.

General format

SYSTEM MENU

TAB Clause

The TAB clause forces the user to end an ACCEPT by typing a valid termination key.

General format

[NO] TAB

General rules

1. The TAB clause is treated as commentary.
2. The NO option is ignored in textual screens; it is active in graphical screens.

TITLE Clause

Set the title property of the screen item component. This is generally the most basic textual element of a graphical component, such as the label text on a push-button.

General format

TITLE [IS|=] title

TO Clause

The TO clause identifies the destination of the data in an ACCEPT screen statement.

General format

{TO|OBJECT} identifier-1

Syntax rules

1. The category of identifier-1 shall be a permissible category as a receiving item in a MOVE statement where the sending item has the same PICTURE clause as the subject of the entry.
2. Identifier-1 shall be defined in the file, working-storage, local-storage, or linkage sections.

General rules

1. The subject of the entry is an input screen item.
2. The TO phrase only has an effect during execution of an ACCEPT screen statement referencing the screen item.

TO Clause Graphical

The TO clause specifies the destination of data during an ACCEPT of an input screen input.

General format

TO [MULTIPLE|TABLE] identifier-1

Syntax rules

1. MULTIPLE and TABLE are synonymous.

General rules

1. Identifier-1 is the destination for the data; this data is read and used for the display of the input item during a DISPLAY.
2. If identifier-1 is a table and the component supports the MULTIPLE type, then MULTIPLE identifier-1 indicates that each element of the table identifier-1 shall be used to fill the component.

TRAILING-SIGN Clause

The TRAILING-SIGN clause is syntax-checked only.

General format

TRAILING-SIGN

General rules

The TRAILING-SIGN clause is syntax-checked only.

UNDERLINE Clause

The UNDERLINE clause specifies that each character of the field is underlined when it is displayed on the screen.

General format

WITH {UNDERLINE|UNDERLINED}

General rules

1. If the UNDERLINE clause is specified at group level, it applies to each elementary screen item in that group.
2. When the UNDERLINE clause is specified, the screen item will be displayed so that the characters that constitute the screen item are underlined.

UPPER Clause

The UPPER clause is syntax-checked only for textual screen items in a graphical screen section.

General format

UPPER

Syntax rules

For textual screens, the UPPER clause is ignored.

USAGE Clause

The USAGE clause specifies the storage and usage format for an elementary item. The default usage is DISPLAY, but this is overridden using the USAGE clause.

All tokens on each line in the memory-usage-token and external-usage-token group are always synonymous; further discussion refers to only the first on each line.

General format

USAGE IS {memory-usage | external-usage | property-object-modifier}

Where memory-usage is one of the following:

{DISPLAY | DISPLAY-1}

{COMP | COMPUTATIONAL}

{COMP-N | COMPUTATIONAL-N}

{COMP-1 | COMPUTATIONAL-1}

{COMP-1-REV | COMPUTATIONAL-1-REV | COMP-1-M | COMPUTATIONAL-1-M}

{COMP-1-MVS | COMPUTATIONAL-1-MVS | COMP-1-E | COMPUTATIONAL-1-E}

{COMP-2 | COMPUTATIONAL-2}

{COMP-2-REV | COMPUTATIONAL-2-REV | COMP-2-M | COMPUTATIONAL-2-M}

{COMP-2-MVS | COMPUTATIONAL-2-MVS | COMP-2-E | COMPUTATIONAL-2-E}

{COMP-3 | COMPUTATIONAL-3}

{COMP-4 | COMPUTATIONAL-4}

{COMP-5 | COMPUTATIONAL-5}

{COMP-6 | COMPUTATIONAL-6}

{COMP-S | COMPUTATIONAL-S | COMP-1-A | COMPUTATIONAL-1-A}

{COMP-D | COMPUTATIONAL-D | COMP-2-A | COMPUTAITONAL-2-A}

{PACKED-DECIMAL | COMP-P | COMPUTATIONAL-P}

{PACKED-DECIMAL-A | COMP-3-A | COMPUTATIONAL-3-A}

{PACKED-DECIMAL-I | PACKED-DECIMAL-E | PACKED-DECIMAL-H | COMP-3-M | COMPUTATIONAL-3-M | PACKED-DECIMAL-M}

{BINARY | BINARY-M | COMP-M | COMPUTATIONAL-M | COMP-A | COMPUTATIONAL-A | COMP-4-M | COMPUTATIONAL-4-M | COMP-B | COMPUTATIONAL-B}

{BINARY-REV}

{SQLIND}

{COMP-X | COMP_X | COMPUTATIONAL-X | COMPUTATIONAL_X | COMP-5-M | COMPUTATIONAL-5-M}

{COMP-X-REV | COMPUTATIONAL-X-REV}

{DISPLAY-WS}

{EXTERNAL-FORM}

{NATIONAL}

{KANJI}

{SIGNED-SHORT}

{UNSIGNED-SHORT}

{SIGNED-INT}

{UNSIGNED-INT}

{SIGNED-LONG}

{UNSIGNED-LONG}

{FLOAT}

{DOUBLE}

{HANDLE OF THREAD}

{HANDLE [OF handle-component [implicit-title]]}

{HANDLE OF FONT [FIXED-FONT|TRADITIONAL-FONT|DEFAULT-FONT| SMALL-FONT|MEDIUM-FONT|LARGE-FONT]}

{HANDLE OF WINDOW [implicit-title]}

{WINDOW}

Where external-usage is one of the following

{INDEX}

{POINTER}

{PROCEDURE-POINTER | PROCEDURE POINTER}

{STRING}

{JPACKED-DECIMAL}
{JSTRING}
{JBYTE}
{JSHORT}
{JINT}
{JLONG}
{JBOOLEAN}
{JFLOAT}
{JDOUBLE}
{JCHAR}
{ALPHANUMERIC-HASHTABLE}
{NUMERIC-HASHTABLE}
{OBJECT-HASHTABLE}
{OBJECT REFERENCE [OF object-classname|classname-literal]}
{OBJECT|COMPONENT [OF object-classname]}

Where property-object-modifier is one of:

PROPERTY basic-type nonnumeric-literal-propname-1 nonnumeric-literal-setval-2
PROPERTY class-name nonnumeric-literal-propname-1 object-reference-setval-2
PROPERTY class-name nonnumeric-literal-propname-1 nonnumeric-literal-setval-2

Where basic-type is one of:

CHARACTER	; Java char
BINARY-BYTE	; Java byte
BINARY-SHORT	; Java short
BINARY	; Java int
BINARY-LONG	; Java int
BINARY-DOUBLE	; Java long
FLOAT-SHORT	; Java float
FLOAT-LONG	; Java double
BIT	; Java Boolean

Syntax rules

1. All tokens on a line are always synonymous.
2. Only the first token on a line will be referred to afterwards in this section.
3. The USAGE clause may be written in any data description or screen description entry with a level-number other than 66 or 88.
4. The PICTURE, if any, must match the USAGE.

General rules

1. A USAGE clause written at the group level applies to its group members.
2. Property object modifiers must be used only in combination with object formats, must follow the object format usage, and are used to set initial properties of the object. The nonnumeric-literal-propname-1 is the property name; if the set method is setProperty, then "property" is used. The setval's are objects or nonnumeric-literals suitable for parameters to the setProperty method. The type must match the given class-name or basic-type. More information is available in the SET verb.

COMP	By default, COMP is the same as PACKED-DECIMAL. Compiler switch -Ci makes it the same as the current COMP-4. Compiler switch -Cr makes it the same as the current COMP-2. Compiler switch -DBR makes it the same as BINARY if already equal to BINARY-REV. Compiler switch -Db or -Db:trunc makes it the same as the current BINARY. Compiler switch -Dca makes it the same as BINARY. Compiler switch -mpe makes it the same as BINARY. Compiler switch -Dcm makes it the same as BINARY. Compiler switch -D2 makes it the same as COMP-D.
COMP-N	By default, COMP-N is the same as COMP-X. Compiler switch -Dca makes it the same as COMP-X-REV on Windows.
COMP-1	Compiler switch -Dca makes it the same as COMP-S. Compiler switch -Dcm makes it the same as COMP-1-REV. Compiler switch -Df makes it the same as COMP-1 (undo previous change). Compiler switch -Cv makes it the same as COMP-1-REV. COMP-1 is an IEEE single-precision floating point number.
COMP-1-REV	COMP-1-REV is an IEEE single-precision floating-point number stored in reverse byte order.
COMP-1-MVS	COMP-1-MVS is a single-precision floating-point number stored in OS/390 MVS format, but calculations are done as an IEEE single-precision floating-point number.
COMP-2	COMP-2 is an IEEE double-precision floating-point number. Compiler switch -Dca makes it the same as COMP-D. Compiler switch -Dcm makes it the same as COMP-2-REV. Compiler switch -Df makes it the same as COMP-2 (undo previous change). Compiler switch -Cv makes it the same as COMP-2-REV.
COMP-2-REV	COMP-2-REVis an IEEE double-precision floating-point number stored in reverse byte order.
COMP-2-MVS	COMP-2-MVS is a double-precision floating-point number stored in OS/390 MVS format, but calculations are done as an IEEE double-precision floating-point number.
COMP-3	By default, COMP-3 is the same as PACKED-DECIMAL. Compiler switch -Dca makes it the same as PACKED-DECIMAL-A. Compiler switch -mpe makes it the same as PACKED-DECIMAL-I. Compiler switch -Dcm makes it the same as PACKED-DECIMAL-I. Compiler switch -D6 makes it the same as COMP-6.
COMP-4	By default, COMP-4 is the same as BINARY. Compiler switch -DBR makes it the same as BINARY-REV. Compiler switch -Dcm makes it the same as BINARY.
COMP-5	By default, COMP-5 is the same as BINARY. Compiler switch -DBR makes it the same as BINARY-REV. Compiler switch -Dca makes it the same as BINARY-REV on Windows. Compiler switch -Dcm makes it the same as COMP-X-REV on Windows, COMP-X off Windows.
COMP-6	COMP-6 is a packed-decimal storage format. This is the AcuCOBOL COMP-6 format.
COMP-S	COMP-S is a 2-byte signed short integer format, -32767 to 32767.
COMP-D	COMP-D is a zoned decimal storage format, storing each digit as its own binary number rather than as an ASCII number.
PACKED-DECIMAL	Compiler switch -Dca makes it the same as PACKED-DECIMAL-A. Compiler switch -mpe makes it the same as PACKED-DECIMAL-I. Compiler switch -Dcm makes it the same as PACKED-DECIMAL-I. Compiler switch -D6 makes it the same as COMP-6. PACKED-DECIMAL stores 2 decimal digits per byte.
PACKED-DECIMAL-A	PACKED-DECIMAL-A stores 2 decimal digits per byte. This is the

	AcuCOBOL PACKED-DECIMAL format.
PACKED-DECIMAL-I	PACKED-DECIMAL-I stores 2 decimal digits per byte. This is the IBM PACKED-DECIMAL format.
BINARY	Compiler switch -DBR makes it the same as BINARY-REV. Compiler switch -Dca makes it the same as COMP-4. Compiler switch -Dcm makes it the same as BINARY (undo previous change). Compiler switch -D5 makes it the same as the current COMP-5. BINARY stores numeric values in binary, base two, form.
BINARY-REV	BINARY-REV stores numeric values in binary, base two, form with bytes reversed.
SQLIND	SQLIND is a SQL indicator type; this is the same as COMP-X PIC S9(9). This type is not required for SQL usage in Elastic COBOL, but is present for compatibility.
COMP-X	Compiler switch -DBR makes it the same as COMP-X-REV. COMP-X stores numeric values in binary, base two, form. The picture may be in PIC X style or PIC 9 style. If PIC X style, the number of X's determines the number of bytes used to store the number (causing binary truncation); if PIC 9 style, the number of bytes allocated will be sufficient to hold the decimal number of 9's given (causing decimal truncation). An 'S' character to indicate sign with PIC X format is supported.
COMP-X-REV	COMP-X-REV stores numeric values in binary, base two, form with bytes reversed. The picture may be in PIC X style or PIC 9 style. If PIC X style, the number of X's determines the number of bytes used to store the number (causing binary truncation); if PIC 9 style, the number of bytes allocated will be sufficient to hold the decimal number of 9's given (causing decimal truncation). An 'S' character to indicate sign with PIC X format is supported.
SIGNED-SHORT	This is the same as COMP-X PIC SX(2) (COMP-X-REV on Windows).
UNSIGNED-SHORT	This is the same as COMP-X PIC X(2) (COMP-X-REV on Windows).
SIGNED-INT	This is the same as COMP-X PIC SX(4) (COMP-X-REV on Windows).
UNSIGNED-INT	This is the same as COMP-X PIC X(4) (COMP-X-REV on Windows).
SIGNED-LONG	Compiler switch -Dca makes it the same as SIGNED-INT. This is the same as COMP-X PIC SX(8) (COMP-X-REV on Windows).
UNSIGNED-LONG	Compiler switch -Dca makes it the same as UNSIGNED-INT. This is the same as COMP-X PIC X(8) (COMP-X-REV on Windows).
FLOAT	Compiler switch -Dca makes it the same as COMP-1-REV. This is the same as COMP-1.
DOUBLE	Compiler switch -Dca makes it the same as COMP-2-REV. This is the same as COMP-2.
DISPLAY	DISPLAY is the default usage, suitable for storing items for DISPLAY to the terminal. A DISPLAY type may be numeric or non-numeric.
DISPLAY-WS	DISPLAY-WS is the same as DISPLAY, but the item is treated as if it were in the SCREEN SECTION even when not. This is useful only for textual screen items with minor amounts of formatting, such as done in Wang COBOL. Avoid this format in new code.
NATIONAL	This is the same as DISPLAY. It serves a documentary purpose when PIC N or PIC G is used.
KANJI	This is the same as DISPLAY. It serves a documentary purpose when PIC N or PIC G is used.

EXTERNAL-FORM	EXTERNAL-FORM marks an item as being an external-form for CGI or Servlet purposes. The IDENTIFIED BY clauses will be used on members of an EXTERNAL-FORM group as the HTML identifiers.
HANDLE OF THREAD HANDLE [OF handle-component [implicit-title]] HANDLE OF FONT [FIXED-FONT| TRADITIONAL-FONT| DEFAULT-FONT| SMALL-FONT| MEDIUM-FONT| LARGE-FONT] HANDLE OF WINDOW [implicit-title]	The HANDLE type is a numeric type, but is used only for storing handles to objects. The object handle must be a graphical object or a thread object, especially designed for use as a handle. A HANDLE may be typed or untyped; if typed, it may be used only for the type of HANDLE specified. If untyped, it may hold any HANDLE type. Typing the HANDLE is important; for graphical MODIFY verbs, only common properties may be used on an untyped HANDLE; properties specific to a type work only on a typed HANDLE. Unless absolutely necessary, specify a type for the HANDLE. The HANDLE OF THREAD may only hold a HANDLE to a THREAD; this is used with PERFORM or CALL THREAD. The HANDLE OF FONT may only hold a FONT; it must be used for any FONT handle operations. There is an extended ability to preload a FONT HANDLE with a particular built-in FONT; in such a case, the build-in FONT is listed after HANDLE OF FONT.
WINDOW	The HANDLE OF WINDOW may only hold a WINDOW; it must be used for any WINDOW handle operations. A graphical HANDLE may have an implicit TITLE property specified after the handle-component type name, such as HANDLE OF PUSH-BUTTON "OK".
INDEX	The INDEX is a traditional, standard usage, used primarily for indexing subscripts into tables.
POINTER	The POINTER usage is an identifier which points to another identifier. The POINTER in Elastic COBOL may only point to Elastic COBOL objects as Java does not allow general pointers. Avoid POINTER usage if possible.
PROCEDURE-POINTER	The PROCEDURE-POINTER usage is an identifier which points to a procedure, an entry-point.
STRING	This is the same as OBJECT REFERENCE "String".
JPACKED-DECIMAL	This is the same as OBJECT REFERENCE "java.math.BigDecimal".
JSTRING	This is the same as OBJECT REFERENCE "String".
JBYTE	This is the Java primitive numeric 'byte', a 1-byte signed numeric. It may only be used where expressly allowed.
JSHORT	This is the Java primitive numeric 'short', a 2-byte signed numeric. It may only be used where expressly allowed.
JINT	This is the Java primitive numeric 'int', a 4-byte signed numeric. It may only be used where expressly allowed.
JLONG	This is the Java primitive numeric 'long', an 8-byte signed numeric. It may only be used where expressly allowed.
JBOOLEAN	This is the Java primitive 'Boolean', which may be true or false. It may only be used where expressly allowed.
JFLOAT	This is the Java primitive numeric 'float', a single-precision IEEE floating-point number. It may only be used where expressly allowed.
JDOUBLE	This is the Java primitive numeric 'double', a double-precision IEEE floating-point number. It may only be used where expressly allowed.
JCHAR	This is the Java primitive numeric 'char', a 2-byte Unicode character. It may only be used where expressly allowed.
ALPHANUMERIC-HASHTABLE	This is a hash table usable directly within Elastic COBOL most anywhere an alphanumeric is allowed; it acts as a keyed array. The SET verb is used to set keys to values or variables. If set to a

	variable, this acts as a named pointer. The usage is: Id_alphanumeric_hashtable ("key") => alphanumeric-id
NUMERIC-HASHTABLE	This is a hash table usable directly within Elastic COBOL most anywhere a numeric is allowed; it acts as a keyed array. The SET verb is used to set keys to values or variables. If set to a variable, this acts as a named pointer. The usage is: Id_numeric_hashtable ("key") => alphanumeric-id
OBJECT-HASHTABLE	This is a hash table usable directly within Elastic COBOL most anywhere an object is allowed; it acts as a keyed array. The SET verb is used to set keys to values or variables. If set to a variable, this acts as a named pointer. The usage is: Id_object_hashtable ("key") => alphanumeric-id
OBJECT| COMPONENT [OF object-classname]	This is the same as OBJECT REFERENCE, except an initial object is already created to fill the reference. This is allowed only for objects with a default constructor (a constructor with no argument parameters).
OBJECT REFERENCE [OF object-classname| classname-literal]	An object reference may be typed or untyped; a typed reference includes the object-classname or classname-literal. If a method requires a certain type, then only a typed reference may be used. An untyped reference may be used as the method to INVOKE, but the INVOKE is then dynamic and not as efficient. Always use a typed OBJECT REFERENCE when possible. The OBJECT REFERENCE is an actual Java Object, with all its capabilities, restrictions, securities, etc. All Java objects ultimately extend java.lang.Object which has certain methods, such as toString(); this is used for the DISPLAY of an OBJECT. A Elastic COBOL program (PROGRAM-ID) is a Java Object and may be referenced as an object, but avoid doing so if possible to avoid syntactic confusion. A Elastic COBOL class object (defined with CLASS-ID) is referenced in the exact same manner as any other Java Object through this syntax. The object-classname is defined in the REPOSITORY paragraph. As an extension, Elastic COBOL supports naming the class inline as a classname-literal. Classname-literal is a nonnumeric-literal exactly specifying the class, such as "java.lang.String". All Java classnames are referred to using dot notation not / notation (java.lang.String, not java/lang/String) and do not include the text '.class' at the end (java.java.String, not java.lang.String.class). Elastic COBOL classes are normally in the default, or anonymous, package and do not have a dot in the name. Unless making an API (Application Programming Interface), generally avoid creating packages in Elastic COBOL applications. The object type validity cannot be checked directly by the Elastic COBOL compiler as it is a native compiler, and the runtime may differ from the compilation environment. When using objects, the Java compiler may give an error if the types are incorrect, or an exception may occur at runtime if the types have changed in a dynamic fashion. The runtime exceptions may be caught using the TRY|CATCH|END-TRY verb. The amount of memory allocated for an OBJECT is determined by the JVM (Java Virtual Machine) and is outside the scope of

	traditional COBOL memory. An OBJECT is always allocated from the Java heap. A single OBJECT may have multiple OBJECT REFERENCES pointing at it. An OBJECT does not need to be tracked; it may be allocated and reallocated without explicitly freeing the previous allocation. Elastic COBOL is running within a garbage-collection environment, where all memory handling is performed automatically. Any time an object has zero references, it is marked as capable of being freed when the heap runs out of memory. Do not allocate a large object and allow it to fall into disuse when an object reference is pointing at it still; set the object reference to null in order to free the object. Garbage collection does not free the programmer from all memory constraints, only from tracking bugs.

USING Clause

The USING clause identifies data to be used both as the destination in an ACCEPT screen statement and the source for a DISPLAY screen statement.

General format

USING identifier-1

Syntax rules

1. The category of identifier-1 shall be a permissible category as a receiving item in a MOVE statement where the sending item has the same PICTURE clause as the subject of the entry.
2. The category of identifier-1 phrase shall be a permissible category as a sending item in a MOVE statement where the receiving item has the same PICTURE clause as the subject of the entry.
3. Identifier-1 shall be defined in the file, working-storage, local-storage, or linkage sections.
4. If the subject of this entry is subject to an OCCURS clause, identifier-1 shall be specified without the subscripting normally required.

General rules

1. Specifying the USING phrase is equivalent to specifying both the TO and FROM phrases, each specifying the same identifier.
2. The subject of the entry is both an input screen item and an output screen item.

USING Clause Graphical

The USING clause specifies the combined source and destination of data during a DISPLAY and ACCEPT of an input screen input.

General format

USING [MULTIPLE|TABLE] identifier-1

Syntax rules

1. MULTIPLE and TABLE are synonymous.

General rules

1. Identifier-1 is the destination for the data; this data is read and used for the display of the input item during a DISPLAY.
2. If identifier-1 is a table and the component supports the MULTIPLE type, then MULTIPLE identifier-1 indicates that each element of the table identifier-1 shall be used to fill the component.

VALUE Clause

The VALUE clause specifies the initial value of communication section, local-storage section, and working-storage section data items, the values associated with condition-names, and the value of report section printable items and screen section displayable items. VALUE settings depend on section type. WORKING-STORAGE section items upon initial program load contain their VALUEs but retain their prior state value when the subprogram is re-entered. Using the INITIAL attribute on the PROGRAM-ID specification will reapply the VALUE clause to WORKING-STORAGE section items upon re-entry. The INITIALIZE statement does not use the VALUE clause to reinitialize items, it sets alphanumeric items to SPACES and numeric items to ZEROES.

General format

Format 1 (data-item):

VALUE IS literal-1

Format 2 (table):

{VALUE|VALUES} [IS|ARE] literal-1

Format 3 (condition-name):

{VALUE|VALUES} [IS|ARE] {literal-2 [{THROUGH|THRU} literal-3}] …

[WHEN SET TO FALSE IS literal-4]

Syntax rules

ALL FORMATS

1. If the category of the item is numeric, all literals in the VALUE clause shall be numeric and have a value that is within the range of values indicated by the PICTURE clause or the USAGE clause. For items with usage other than float-short, float-long and float-extended, this value shall not require truncation of non-zero digits.

   NOTE - if the usage of the item is floating point, the actual value given to the item is subject to standard floating point imprecision, and shall be an approximation of the arithmetic value of the literal.
2. A signed numeric literal shall have associated with it a signed numeric PICTURE character-string.
3. If the category of the item is alphabetic, alphanumeric, alphanumeric-edited, or numeric-edited, literals in the VALUE clause shall be alphanumeric literals.

Editing characters in the PICTURE clause are included in determining the size of the data item but have no effect on initialization of the data item. Alphanumeric literals in the VALUE clause of an elementary item shall not exceed the size indicated by an explicit PICTURE clause. Alphanumeric literals in the VALUE clause of a group item shall not exceed the size of the group item.

4. If the category of the item is national or national-edited, literals in the VALUE clause shall be national literals. These national literals shall not exceed the size indicated by an explicit PICTURE clause.
5. Editing characters in a PICTURE character-string are included in determining the size of the data item but have no effect on initialization of the data item.

NOTE - The programmer is responsible for specifying the value of a literal associated with an alphanumeric-edited, numeric-edited, or national-edited item in edited form.

FORMATS 2 AND 3

The words VALUE and VALUES are equivalent.

FORMATS 1 AND 2

1. Formats 1 and 2 shall not be specified in a condition-name entry.
2. The VALUE clause shall not be specified in a data description entry that contains a REDEFINES clause or in an entry that is subordinate to an entry containing a REDEFINES clause.
3. If the VALUE clause is specified at the group level, literal-1 shall be a figurative constant or an alphanumeric literal. The VALUE clause shall not be specified at subordinate levels within this group.
4. The VALUE clause shall not be specified for a group item containing items subordinate to it with descriptions including JUSTIFIED, SYNCHRONIZED, or a USAGE other than usage display.

FORMAT 1

In the screen section, literal-1 shall be an alphanumeric literal that is not a figurative constant.

FORMAT 2

A data description entry that contains the VALUE clause shall contain an OCCURS clause or be subordinate to a data description entry that contains an OCCURS clause.

FORMAT 3

1. The words THROUGH and THRU are equivalent.
2. Literal-2 shall be less than literal-3.
3. Literal-4 shall not be equal to any literal-2, and in any literal-2 THROUGH literal-3 pair, literal-4 shall not be greater than or equal to literal-2 and less than or equal to literal-3.

General rules

FORMATS 1 AND 2

1. In the file section, VALUE clauses take effect only when the program is placed into its initial state such as when it is first entered and, for WORKING-STORAGE section items, when it is re-entered if the subprogram contains the INITIAL attribute on the PROGRAM-ID specification. The initial value of the data items in the file section is undefined.
   The initial value of data items in the file section is defined to be the same as that of data items in the working-storage section.
2. In the linkage section, VALUE clauses take effect only when the program is placed into its initial state and repeatedly upon being re-entered if the subprogram contains the INITIAL attribute on the PROGRAM-ID specification.
3. In the working-storage, local-storage, and communication sections, VALUE clauses take effect only when the program is placed into its initial state. If the VALUE clause is specified in the description of the data item, the data item is initialized to the defined value. If the VALUE clause is not associated with a data item, the initial value of that data item is undefined, with the exception of data items of category object reference and data items of class pointer.
4. A data item of class object is initialized to reference the NULL object. This initial value takes effect any time a VALUE clause takes effect.
5. A data item of class pointer is initialized to the predefined address NULL. This initial value takes effect any time a VALUE clause takes effect.
6. A VALUE clause specified in the data description entry of a based item, or a data description entry subordinate to a based itemwhen the program is placed into its initial state.
7. A VALUE clause specified in a record description entry containing the EXTERNAL clause takes effect only when the program is placed into its initial state.
8. If a VALUE clause is specified in a data description entry of a group item, the group area is initialized without consideration for the individual elementary or group items contained within this group.
9. If a VALUE clause is specified in a data description entry of a data item that is associated with a variable-occurrence data item, the initialization of the data item behaves as if the value of the data item referenced by the DEPENDING ON phrase in the OCCURS clause specified for the variable-occurrence data item is set to the maximum number of occurrences as specified by that OCCURS clause. A data item is associated with a variable-occurrence data item in any of the following cases:
   a. It is a group data item that contains a variable-occurrence data item.
   b. It is a variable-occurrence data item.
   c. It is a data item that is subordinate to a variable-occurrence data item.

If a VALUE clause is associated with the data item referenced by a DEPENDING ON phrase, that value is considered to be placed in the data item after the variable-occurrence data item is initialized. (See OCCURS clause.)

FORMATS 1 AND 2

1. Each literal is aligned in the associated data item in accordance with 13.2.8, Standard alignment rules, except that initialization is not affected by a JUSTIFIED clause and no editing takes place.
2. Initialization is not affected by any BLANK WHEN ZERO clause that may be specified.

FORMAT 1

A VALUE clause specified in a data description entry that contains an OCCURS clause or in an entry that is subordinate to an OCCURS clause causes every occurrence of the associated data item to be assigned the specified value.

FORMAT 1

If a VALUE clause is specified for a screen item, literal-1 specifies the value of the screen item that is displayed on the screen when directly or indirectly referenced by a DISPLAY screen statement.

FORMAT 2

1. A format 2 VALUE clause initializes a table element to the value of literal-1.

FORMAT 3

1. The characteristics of a condition-name are implicitly those of its conditional variable.
2. The FALSE phrase has meaning only if the associated condition-name is referenced in a SET condition-name TO FALSE statement.

VALUE Clause Graphical

Specify the value source for a graphical component. This corresponds to the value property for the component if present.

General format

VALUE [MULTIPLE|TABLE] from-item

Syntax rules

1. The from-item may be a table of multiple items for certain components, when used with the MULTIPLE or TABLE descriptor.
2. MULTIPLE and TABLE are synonymous.

VISIBLE Clause

Set the visibility for a graphical component.

General format

VISIBLE [=] value

Syntax rules

VISIBLE is ignored for textual screens.

General rules

VISIBLE determines if a graphical component is visible to the user if the parent window is also visible.

ZERO-FILL Clause

The ZERO-FILL clause causes data in free-format non-edited numeric data items to appear on the screen with no zero-suppression in all integer character positions.

General format

WITH ZERO-FILL

General rules

1. This clause is applicable only to numeric non-edited data items.
2. This clause is effective when initial data is displayed and when an ACCEPT into the data item is terminated.
3. This clause is treated as commentary.

13. Procedure Division

The procedure division in a program or method contains procedures to be executed.

General format

Format 1 (with-sections):

```
PROCEDURE DIVISION [using-phrase] .
[       DECLARATIVES.
        { section-name SECTION.
                use-statement.
        [paragraph-name. [sentence]… ] … } …
        END DECLARATIVES. ]
{ section-name SECTION.
        [paragraph-name. [sentence]… ] … } …
```

Where assertion-phrase is:

```
[ASSERT {ENTER|ENTRY} condition-1]…
[ASSERT {EXIT} condition-2]…
```

Where using-phrase is:

```
{USING|CHAINING}
{       [[BY REFERENCE] data-name-1…]
| [{BY CONTENT} data-name-1…]
| [{BY VALUE} data-name-1…]
}… [{GIVING|RETURNING} INTO data-name-2]
```

Format 2 (without-sections):

```
PROCEDURE DIVISION [using-phrase].
{paragraph-name. [SYNCHRONIZED][FULL] |BUILD]}
```

Where assertion-phrase is:

```
[ASSERT {ENTER|ENTRY} condition-1]…
[ASSERT {EXIT} condition-2]…
```

Where using-phrase is:

```
{USING|CHAINING}
{       [[BY REFERENCE] data-name-1…]
| [{BY CONTENT} data-name-1…]
| [{BY VALUE} data-name-1…]
}… [{GIVING|RETURNING} INTO data-name-2]
```

Syntax rules

1. Data-name-1 shall be defined as a level 01 entry or a level 77 entry in the linkage section. A particular user-defined word shall not appear more than once as data-name-1. The data description entry for data-name-1 shall not contain a REDEFINES clause. A data item elsewhere in the linkage section may specify REDEFINES data-name-1.
2. Each data-name-1 specified shall be defined as a data item of class alphanumeric, numeric or object.

3. Data-name-2 shall be specified no more than once, defining an identifier as the return value for the program or method.
4. Condition-1 shall be valid upon entry into the program.
5. Condition-2 shall be valid upon exit from the program.

General rules

1. Execution begins with the first statement of the procedure division, excluding declaratives. Statements are then executed in the order in which they are presented for compilation, except where the rules indicate some other order.
2. The USING phrase identifies the names of the formal parameters used by the method, or program for any arguments passed to it. The arguments passed to it are identified in the activating source element by one of the following:

   } the USING phrase of a CALL statement

   The correspondence between the two lists of names is established on a positional basis.
3. Data-name-1 is a formal parameter for the program.
4. If the argument is passed by reference, the activated runtime entity operates as if the formal parameter occupies the same storage area as the argument.
5. If the argument is passed by content, the activated run time entity operates as if the record in the linkage section were allocated by the activating run time entity during the process of initiating the activation and as if this record does not occupy the same storage area as the argument in the activating run time entity.

   This allocated record is exactly the same number of alphanumeric character positions in length as the argument. That argument is moved to this allocated record without conversion. This record is then treated by the activated run time entity as if it were the argument and as if it were passed by reference.
6. If the argument is passed by value, the activated run time entity operates as if the record in the linkage section were allocated by the activating run time entity during the process of initiating the activation.

   The allocated record is exactly the same number of alphanumeric character positions in length as the storage area in the linkage section. The argument is used as the sending operand and the allocated record, whose description is specified in the linkage section, is used as the receiving operand in the following:

   - if the formal parameter is numeric, a COMPUTE statement without the ROUNDED phrase
   - if the formal parameter is of class object or pointer, a SET statement.

   The activated run time entity is given access to the allocated record.
7. At all times in the activated element, references to data-name-1 and to data-name-2 are resolved in accordance with their description in the linkage section.
8. When either the activating or the activated run time entity is other than a COBOL entity, additional restrictions may apply.

9. Calls from Java follow the definitions given for the callableProgram interface.
10. Condition-1 represents a pre-condition assertion. This must always be true upon entering the program. It serves as documentation only unless the compiler has assertion checking enabled using the '-assert' compiler switch. If assertions are enabled, then an assertion exception is thrown with a message. If this is the outermost called program, then a message will print upon failure; if this is not the outermost program, the calling program's ON EXCEPTION clause will be activated.
11. Condition-2 represents a post-condition assertion. This must always be true upon exiting the program. It serves as documentation only unless the compiler has assertion checking enabled using the '-assert' compiler switch. If assertions are enabled, then an assertion exception is thrown with a message. If this is the outermost called program, then a message will print upon failure; if this is not the outermost program, the calling program's ON EXCEPTION clause will be activated.

Declaratives

A Declarative is a procedure that is to be executed when a specific exception or condition occurs based on the USE statement. Declarative sections shall be grouped at the beginning of the procedure division preceded by the key word DECLARATIVES and followed by the key words END DECLARATIVES.

Procedures

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

Sections

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

Paragraphs

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

Statements and sentences

A statement is a unit of the COBOL language in the procedure division that specifies an action to be taken by a program or method.

The types of statements are conditional; declarative; imperative; and delimited scope, a form of imperative statement.

The types of sentences are conditional, declarative, imperative, and source text manipulation.

Conditional statements and sentences

Conditional statement

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

A conditional statement is one of the following:

- An EVALUATE or IF statement.
- A statement for which a conditional phrase is specified in the source unit.

Conditional phrase

A conditional phrase specifies the action to be taken upon determination of the truth-value of a condition resulting from the execution of a conditional statement. A conditional phrase is one of the following:

1. AT END or NOT AT END phrase when specified within a READ, RETURN, or SEARCH statement.
2. INVALID KEY or NOT INVALID KEY phrase when specified within a DELETE, READ, REWRITE, START, or WRITE statement.
3. END-OF-PAGE or NOT END-OF-PAGE phrase when specified within a WRITE statement.
4. SIZE ERROR or NOT ON SIZE ERROR phrase when specified within an ADD, COMPUTE, DIVIDE, MULTIPLY, or SUBTRACT statement.
5. ON OVERFLOW or NOT ON OVERFLOW phrase when specified within a STRING or UNSTRING statement.
6. ON OVERFLOW, ON EXCEPTION, or NOT ON EXCEPTION phrase when specified within a CALL statement.
7. ON EXCEPTION or NOT ON EXCEPTION phrase when specified within an ACCEPT, DISPLAY, INVOKE or other statement.

Conditional sentence

A conditional sentence is a conditional statement, optionally preceded by an imperative statement, terminated by the separator period.

Declarative statements and declarative sentences

Declarative statement

A declarative statement begins with the word USE and directs the compiler to generate code to take specific action during the processing of other statements.

Declarative sentence

A declarative sentence is a declarative statement terminated by a separator period.

Imperative statements and imperative sentences

Imperative statement

An imperative statement specifies an unconditional action to be taken by the object program or is a conditional statement that is delimited by its explicit scope terminator (delimited scope statement). An imperative statement may consist of a sequence of imperative statements.

Imperative statement names

ACCEPT[8]	FREE	RESUME
ADD[1]	GOBACK	REWRITE[2]
ALLOCATE	GO TO	SEND
ALTER	INITIALIZE	SET
CALL[7]	INSPECT	SORT
CANCEL	INVOKE[8]	START[2]
CLOSE	MERGE	STOP
COMPUTE[1]	MOVE	STRING[3]
CONTINUE	MULTIPLY[1]	SUBTRACT[1]
DELETE[2]	OPEN	SUPPRESS
DISPLAY[8]	PERFORM	UNLOCK
DIVIDE[1]	PURGE	UNSTRING[3]
EXCLUSIVE	READ[5]	WRITE[6]
EXEC	RECEIVE[4]	
EXIT	RELEASE	

1 Without the optional ON SIZE ERROR and NOT ON SIZE ERROR phrases
2 Without the optional INVALID KEY and NOT INVALID KEY phrases
3 Without the optional ON OVERFLOW and NOT ON OVERFLOW phrases
4 Without the optional NO DATA and WITH DATA phrases
5 Without the optional AT END, NOT AT END, INVALID KEY, and NOT INVALID KEY phrases
6 Without the optional INVALID KEY, NOT INVALID KEY, END-OF-PAGE, and NOT END-OF-PAGE phrases
7 Without the optional ON OVERFLOW, ON EXCEPTION, and NOT ON EXCEPTION phrases
8 Without the optional ON EXCEPTION or NOT ON EXCEPTION phrases

Wherever 'imperative-statement' appears in the general format of statements, 'imperative-statement' refers to that sequence of consecutive imperative statements that shall be ended by a period or by any phrase associated with a statement containing that 'imperative-statement'.

Imperative sentence

An imperative sentence is an imperative statement terminated by the separator period.

Delimited scope statements

A delimited scope statement is any statement that includes its explicit scope terminator.

Scope of statements

Scope terminators delimit the scope of certain procedure division statements. Statements that include their explicit scope terminators are termed delimited scope statements. The scope of statements that are contained within statements (nested) may also be implicitly terminated.

When statements are nested within other statements, a separator period that terminates the sentence also implicitly terminates all nested statements. Whenever any statement is contained within another statement, the next phrase of the containing statement following the contained statement terminates the scope of any unterminated contained statement.

When a delimited scope statement is nested within another delimited scope statement with the same statement name, each explicit scope terminator terminates the statement begun by the most recently preceding, and as yet unterminated, occurrence of that statement name.

When statements are nested within other statements that allow optional conditional phrases, any optional conditional phrase encountered is considered to be the next phrase of the nearest preceding unterminated statement with which that phrase is permitted to be associated according to the general format and the syntax rules for that statement, but with which no such phrase has already been associated. An unterminated statement is one that has not been previously terminated either explicitly or implicitly.

Explicit and implicit scope terminators

Scope terminators serve to delimit the scope of certain procedure division statements. Scope terminators are of two types: explicit and implicit.

Explicit scope terminators

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

The implicit scope terminators are the following:

1. At the end of any sentence, the separator period that terminates the scope of all previous statements not yet terminated.
2. Within any statement containing another statement, the next phrase of the containing statement following the contained statement terminates the scope of any unterminated contained statement.

Execution

State of a function, method, object, or program

State of a function, method, or a program

The state of a function, method, or program at any point in time in a run unit may be active or inactive. When a program is activated, its state may also be initial or last-used. Functions and methods are in the initial state every time they are activated.

Active state

A function, method, or program may be activated recursively. Therefore, several instances of a function, method, or program may be active at once.

An instance of a function is placed in an active state when it is successfully activated and remains active until the execution of a STOP statement or an implicit or explicit function format of the EXIT statement within this instance of this function.

An instance of a method is placed in an active state when it is successfully activated and remains active until the execution of a STOP statement or implicit or explicit method format of the EXIT statement within this instance of this method.

An instance of a program is placed in an active state when it is successfully activated by the operating system or successfully called from a run time entity. An instance of a program remains active until the execution of one of the following:

- a STOP statement
- in a called program, an implicit or explicit program format of an EXIT statement within that program
- a GOBACK statement within either that same called program or a program that is not under the control of a calling run time entity.

Whenever a function, method, or program is activated, the control mechanisms for all PERFORM statements contained in the function, method, or program are set to their initial states and the initial function, method, or program collating sequences are in effect.

Initial and last-used states of data

When a function, method, or program is activated, the data within is in either the initial state or the last-used state.

Initial state

Automatic data and initial data are placed in the initial state every time the function, method, or program in which it is described is activated.

Static data is placed in the initial state:

1. The first time the function, method, or program in which it is described is activated in a run unit.
2. The first time the program in which it is described is activated after the execution of an activating statement referencing a program that possesses the initial attribute and directly or indirectly contains the program.
3. The first time the program in which it is described is activated after the execution of a CANCEL statement referencing the program or CANCEL statement referencing a program that directly or indirectly contains the program. This may be negated by use of the RESIDENT clause.

When data in a function, method, or program is placed in the initial state, the following occurs:

1. The internal data described in the working-storage section and local-storage section is initialized in the following way: if a VALUE clause is used in the description of a data item, the data item is initialized to the defined value, or if a VALUE clause is not associated with a data item, the initial value of the data item is undefined. The initial value of an index is undefined.
2. The program's internal file connectors are initialized by setting them to not be in any open mode.

Last-used state

Static and external data are the only data that are in the last-used state. External data is always in the last-used state except when the run unit is activated. Static data is in the last-used state except when it is in the initial state as defined above.

Explicit and implicit transfers of control

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

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

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

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

NOTE - Another implicit transfer of control occurs after execution of the declarative section, as described in rule 1 above.

An explicit transfer of control consists of an alteration of the implicit control transfer mechanism by the execution of a procedure branching or conditional statement. An explicit transfer of control may be caused only by the execution of a procedure branching or conditional statement. The procedure branching statement EXIT PROGRAM causes an explicit transfer of control only when the statement is executed in a called program.

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

There is no next executable statement when the program contains no procedure division or following:

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

2. The last statement in a declarative section when the statement is in the range of an active PERFORM statement executed in a different section and this last statement of the declarative section is not also the last statement of the procedure that is the exit of the active PERFORM statement.
3. The last statement in a source element when the paragraph in which it appears is not being executed under the control of some other COBOL statement in that source element.
4. A STOP RUN statement or EXIT PROGRAM statement that transfers control outside the COBOL source element.
5. The end marker.

When there is no next executable statement and control is not transferred outside the source unit, the flow of control is undefined unless

1. in a program, execution is in the non-declarative procedures portion and the program is under the control of a CALL statement, in which case an implicit EXIT PROGRAM statement is executed.

Item identification

Item identification is the process of identifying a specific data item referenced by an identifier by evaluating all of the identifier's references. The evaluation proceeds in the following order:

1. If the identifier is a function identifier, the function is evaluated according to the rules for evaluating function identifiers.
2. If the identifier is subscripted, all subscripts are evaluated according to the rules for the evaluation of subscripts given in Subscripts.
3. If the data item referenced by the identifier is a group data item that contains a variable-occurrence data item, the length of the data item is evaluated according to the rules for the OCCURS clause.
4. If the identifier is reference-modified, the reference modifiers are evaluated according to the rules for the evaluation of reference modifiers given in Reference-modifier.

Unless otherwise specified by the rules for a statement, any item identification is done only once as the first operation of the execution of that statement.

Sending and Receiving operands

An operand is a sending operand if its contents prior to the execution of a statement may be used by the execution of the statement. An operand is a receiving operand if its contents may be changed by the execution of the statement. Operands may be referenced either explicitly or implicitly by a statement. For some statements, an operand is both a sending operand and a receiving operand. The rules for a statement specify whether operands are sending operands, receiving operands, or both when this is not clear from the context of the statement.

Run unit termination

When the run unit is terminated, any storage obtained with an ALLOCATE statement and not previously released with a FREE statement is released.

Overlapping operands

When a sending and a receiving data 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 undefined. For statements in which the sending and receiving data items are defined by the same data description entry, the results of the execution of the statement may or may not be defined depending on the general rules associated with the applicable statement. If there are no specific rules addressing such overlapping operands, the results are undefined.

In the case of reference modification, the unique data item produced by reference modification is not considered to be the same data description entry as any other data description entry. Therefore, if an overlapping situation exists, the results of the operation are undefined.

Multiple results in arithmetic statements

The ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements may have multiple resultant identifiers. These statements are executed in the following order:

1. The initial evaluation of the statement is done and the result of this operation is placed in an intermediate data item. The rules indicating which data items or literals is part of this evaluation are given in the rules for the individual statements. All item identification for the data items involved in the initial evaluation is done at the start of the execution of the statement.
2. The intermediate data item is stored in or combined with and then stored in each single resulting data item in the left-to-right order in which the receiving data items are specified in the statement. Item identification for the receiving data items is done as each data item is accessed unless it was already done in step 1.

Condition handling

Fatal exception conditions

When a fatal exception condition is raised, one of the following occurs in the order specified:

1. If the statement in which the exception was raised is specified with an exception phrase without the NOT phrase and the rules for that statement indicate that the fatal exception condition is to be processed by the exception phrase, the

procedures for non-fatal exceptions as specified in Non-fatal exception conditions apply.

NOTE - Examples are the ON SIZE ERROR phrase and the ON EXCEPTION phrase in CALL.

2. Otherwise, execution of the run unit is terminated abnormally.

Non-fatal exception conditions

If a non-fatal exception condition is raised, one of the following occurs in the order specified:

1. If an exception phrase without the NOT phrase is specified in the statement in which the exception condition is raised, the imperative statement associated with that exception phrase is executed as specified in the rules for the specific statement. The exception phrases are:
   a. AT END in READ, RETURN, and SEARCH,
   b. ON SIZE ERROR in ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT,
   c. INVALID KEY in DELETE, READ, REWRITE, START, and WRITE.
   d. ON OVERFLOW in CALL, STRING, and UNSTRING, and
   e. ON EXCEPTION in ACCEPT, CALL, DISPLAY, or INVOKE.
2. Otherwise, execution of the statement continues as specified in the rules for that statement.

Common phrases and features for statements

This clause provides a description of the common phrases and features that pertain to or appear in several different statements.

ROUNDED phrase

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

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

ON SIZE ERROR phrase and size error condition

The SIZE ERROR and NOT SIZE ERROR phrases are used to check for the size error condition being raised during the execution of an arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT).

When the SIZE ERROR or NOT SIZE ERROR phrase is specified for an arithmetic statement, checking for the size error condition is enabled for the arithmetic operations that take place in developing and storing the result of that arithmetic statement.

The size error condition is raised in the following ways:

1. If the rules for evaluation of exponentiation are violated, the size error condition is raised.
2. If the divisor in a divide operation or the DIVIDE statement is zero, the size error condition is raised.
3. If, after radix point alignment, the absolute value of the result of an arithmetic statement exceeds the largest value that may be contained in the associated resultant data item, the size error condition is raised. The largest value that may be contained in the resultant data item is the maximum value implied by the associated PICTURE character-string, or beyond bounds specified by USAGE for those items that are specified without the PICTURE clause. If the ROUNDED phrase is specified, rounding takes place before checking for size error.

If the size error condition is raised, the following occurs:

1. If the size error condition was raised during the arithmetic operations specified by the arithmetic statement, the values of all of the resultant data items remain unchanged from the values they had at the start of the execution of the arithmetic statement. Execution proceeds as indicated in rule 3, below.
2. If the absolute value of the result of the arithmetic operation exceeds the maximum value allowed for any resultant identifier, the content of that resultant identifier is not changed from the content that existed at the start of the execution of the arithmetic statement. The values of resultant identifiers for which the size error condition was not raised are the same as they would have been if the size error condition had not been raised for any of the resultant identifiers. Execution proceeds as indicated in rule 3, below.
3. After completion of the arithmetic operations, and possibly the storing of values into resultant data items as specified in rules 1 and 2, the following occurs:
    a. If the SIZE ERROR phrase is specified, control is transferred to the imperative-statement specified in the SIZE ERROR phrase and execution continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the SIZE ERROR phrase, control is transferred to the end of the arithmetic statement and the NOT SIZE ERROR phrase, if specified, is ignored.
    b. If the SIZE ERROR phrase is not specified but the NOT SIZE ERROR is specified, control is transferred to the end of the arithmetic statement and the NOT SIZE ERROR phrase is ignored.

    If no size error condition was raised during the execution of the arithmetic operations specified by an arithmetic statement or while storing into the resultant identifiers, the SIZE ERROR phrase, if specified, is ignored and

control is transferred to the end of the arithmetic statement or to the imperative-statement specified in the NOT SIZE ERROR phrase if it is specified. In the latter case, execution continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the NOT SIZE ERROR phrase, control is transferred to the end of the arithmetic statement.

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

CORRESPONDING phrase

For the purpose of this discussion, D1 and D2 are identifiers that refer to group items. A pair of data items corresponds if:

1. A data item in D1 and a data item in D2 are not implicitly or explicitly described with the key word FILLER and have the same data-name and the same qualifiers, if any, up to, but not including, D1 and D2.
2. In a MOVE statement, at least one of the data items is an elementary data item and the resulting move is valid according to the rules for the MOVE statement.
3. In an ADD or SUBTRACT statement, both of the data items are numeric data items.
4. Neither data item contains an OCCURS, REDEFINES, or RENAMES clause or is of class index, object, or pointer.
5. Neither data item is subordinate to a group item that is subordinate to D1 or D2 and the group item contains an OCCURS or REDEFINES clause.
6. The name of each data item that satisfies the above conditions is unique after application of the implied qualifiers.

Any item identification associated with a corresponding pair of operands is done at the start of the execution of the statement containing the CORRESPONDING phrase, not at the start of the implied statement used for the pair of operands. The implied statements are executed in the order in which the elements in the group data item immediately following CORRESPONDING are specified.

Arithmetic statements

The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements. They have several common features.

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

2. The maximum size of each operand shall be 18 digits.
3. For the ADD, DIVIDE, MULTIPLY, and SUBTRACT statements, the composite of operands shall not contain more than 18 digits when none of the operands is an intrinsic function. When any of the operands is an intrinsic function, the composite of the operands that are not intrinsic functions shall not contain more than 18 digits. The composite of operands shall be a hypothetical data item resulting from the superimposition of specified operands in a statement aligned on their decimal points.

Conformance for parameters

Parameters

The number of arguments in the activating unit shall be equal to the number of formal parameters in the activated unit.

The rules for conformance between arguments and formal parameters depend on whether either the formal parameter or its corresponding argument is a group item or both are elementary items.

Group items

If either the formal parameter or the argument is a group item, the corresponding argument or formal parameter shall be a group item or an elementary item of category alphanumeric, and the formal parameter shall be described with the same number or a smaller number of character positions as the corresponding argument.

Elementary items

The conformance rules for elementary items depend on whether the argument is passed by reference, by content, or by value.

Elementary items passed by reference

If neither the formal parameter nor the argument is of class object or pointer, the conformance rules are the following:

1. The formal parameter shall be described with the same number of character positions as the corresponding argument.

Elementary items passed by content or by value

If the formal parameter is not of class object or pointer, the conformance rules are the following:

1. The formal parameter shall be described with the same number of character positions as the corresponding argument.

ACCEPT Statement

The ACCEPT statement causes data to be made available to the specified data item.

The ACCEPT statement causes data associated with a screen item and entered or modified by the operator of a terminal at a specified position and with specified attributes on that terminal, to be placed in the specified data item.

General format

Format 1 (hardware):

ACCEPT identifier-1 [FROM mnemonic-display-device] [END-ACCEPT]

Format 2 (chronological):

```
ACCEPT identifier-1 FROM {      DATE |
                                DAY |
                                DAY-OF-WEEK |
                                TIME |
                                YEAR |
                                DATE {YYYYMMDD|CENTURY-DATE} |
                                DAY {YYYYDDD|CENTURY-DAY} }
    [END-ACCEPT]
```

Format 3 (screen):

```
ACCEPT {identifier-1|screen-name-1} {accept-screen-options}…
[ON EXCEPTION imperative-statement-1]
[NOT ON EXCEPTION imperative-statement-2]
[END-ACCEPT]
```

Format 4 (intrinsic function):

ACCEPT identifier-1 FROM intrinsic-function

Format 5 (various):

Where accept-screen-option is one of the following:

WITH [NO] ADVANCING

WITH { AUTO | AUTO-SKIP | AUTOTERMINATE | AUTOMATIC }

WITH { BACKGROUND-COLOR | BACKGROUND-COLOUR } color-name

WITH { FOREGROUND-COLOR | FOREGROUND-COLOUR } color-name

WITH [NO] { BELL | BEEP }

WITH [CONVERT | CONVERSION]

[NO] TAB

WITH BLANK [SCREEN | LINE | EOL | TO END OF LINE | EOS | TO END OF SCREEN]

WITH ERASE [SCREEN | EOL | TO END OF LINE | EOS | TO END OF SCREEN]

WITH { BLINK | BLINKING }

TRAILING-SIGN

{AT | FROM} COLUMN NUMBER [PLUS | + | -] column-number [CELLS | PIXELS]

{AT | FROM} COLUMN NUMBER [PLUS | + | -] ccolumn-number [CELLS | PIXELS]

{AT | FROM} LINE NUMBER [PLUS | + | -] line-number [CELLS | PIXELS]

{AT | FROM} CLINE NUMBER [PLUS | + | -] cline-number [CELLS | PIXELS]

WITH FULL

WITH {HIGHLIGHT | BRIGHT | HIGH} | {NO {LOWLIGHT | DIM | LOW}}
WITH {LOWLIGHT} | {NO HIGHLIGHT}
WITH { REQUIRED | EMPTY-CHECK }
WITH {REVERSE-VIDEO | REVERSE | REVERSED}
WITH {SECURE} | {NO ECHO} | {OFF} | {NO-ECHO}
WITH { UNDERLINE | UNDERLINED }
WITH SPACE-FILL
WITH ZERO-FILL
WITH OVERLINE
WITH LEFTLINE
WITH GRIDLINE
AT at-value
WITH PROTECTED SIZE size-value
WITH PROMPT CHARACTER prompt
WITH NO PROMPT
WITH [NO] UPDATE
WITH COLOR color
ONLY PFKEYS pfkey…
FOR for-name
UNTIL accept-condition
CONTROL KEY IN control-key-identifier
VALUE IN identifier-1
VALUE OF {MULTIPLE | TABLE} IN identifier-1
FROM display-mnemonic-name

Format 6 (user name)
ACCEPT identifier-1 FROM USER NAME

Format 8 (escape key)
ACCEPT identifier-1 FROM ESCAPE KEY

Format 9 (external line number):
ACCEPT identifier-1 FROM EXTERNAL LINE NUMBER

Format 10 (external column number):
ACCEPT identifier-1 FROM EXTERNAL COLUMN NUMBER

Format 11 (terminal-info):
ACCEPT identifier-1 FROM TERMINAL-INFO

Format 12 (system-info):
ACCEPT identifier-1 FROM SYSTEM-INFO

Format 14 (standard object, GfxScreen):
ACCEPT handle-1 FROM STANDARD OBJECT standard-object

Format 15 (font object, GfxScreen):
ACCEPT handle-1 FROM FONT OBJECT font-name

Format 16 (window handle, GfxScreen):
ACCEPT handle-1 FROM WINDOW HANDLE

Format 17 (handle, GfxScreen):
ACCEPT {handle-1 | identifier-1} [accept-screen-options]

Format 18 (key):
ACCEPT identifier-1 FROM KEY mnemonic-display-device

Format 19 (environment):
ACCEPT identifier-1 FROM ENVIRONMENT env-name

Format 20 (configuration):
ACCEPT identifier-1 FROM CONFIGURATION conf-name

Format 22 (free):
ACCEPT identifier-1 FREE [mnemonic-display-device]

Format 23 (omitted, GfxScreen):
ACCEPT OMITTED [accept-screen-options]

Format 24 (input box, GfxScreen):
ACCEPT INPUT display-parameter [TITLE title] [{GIVING | RETURNING} INTO identifier]

Syntax rules

1. Size-value is an integer or content of integer identifier.
2. At-value is an integer or the contents of an integer identifier. The integer must be 2, 4, 6 or 8 digits in length, divided equally between line and column.
3. Column-number is a numeric or contents of numeric identifier; integer values are used for text. It represents the starting column number, and it may be relative to the current column number.
4. Line-number is a numeric or contents of numeric identifier; integer values are used for text. It represents the starting line number, and it may be relative to the current line number.
5. Ccolumn-number is the character-based column number; Elastic COBOL displays graphically on X Windows so this value is ignored.
6. Cline-number is the character-based line number; Elastic COBOL displays graphically on X Windows so this value is ignored.
7. Prompt is a character literal or the first character of a nonnumeric identifier, or an integer literal; NO PROMPT represents a space prompt.
8. Color is a combined background and foreground-color value, an integer or the contents of an integer identifier.
9. Pfkey is used for Wang compatibility; it is an integer or the contents of an integer identifier.
10. For-name is for AS/400 compatibility; it must be at most 10 characters long.
11. Library-name is for AS/400 compatibility; it must be at most 10 characters long.
12. Display-mnemonic name is a display device.

General rules

ALL FORMATS

The END-ACCEPT phrase delimits the scope of the ACCEPT statement. (See Scope of statements.)

FORMAT 1

1. The ACCEPT statement causes the transfer of data from the hardware device. This data replaces the content of the data item referenced by identifier-1.
2. If a hardware device is capable of transferring data of the same size as the receiving data item, the transferred data is stored in the receiving data item.
3. If a hardware device is not capable of transferring data of the same size as the receiving data item, then:
    a. If the size of the receiving data item (or of the portion of the receiving data item not yet currently occupied by transferred data) exceeds the size of the transferred data, the transferred data is stored aligned to the left in the receiving data item (or the portion of the receiving data item not yet occupied), and additional data is requested.
    b. If the size of the transferred data exceeds the size of the receiving data item (or the portion of the receiving data item not yet occupied by transferred data), only the leftmost characters of the transferred data are stored in the receiving data item (or in the portion remaining). The remaining characters of the transferred data that do not fit into the receiving data item are ignored.
4. The default device used is CONSOLE unless otherwise specified to be SYSIN by use of the compiler option defsys.
5. Control-key-identifier specifies the identifier to hold the control key which terminates the accept, if applicable.

FORMAT 2

1. The ACCEPT statement causes the information requested to be transferred to the data item specified by identifier-2 according to the rules for the MOVE statement. DATE, DAY, DAY-OF-WEEK, YEAR, and TIME reference the current data and time provided by the system on which the ACCEPT statement is executed. DATE, DAY, DAY-OF-WEEK, and TIME are conceptual data items and, therefore, are not described in the COBOL program.
2. DATE, without the phrase YYYYMMDD, is composed of the data elements: two lower-order digits of the year in the Gregorian calendar, month of the year, and day of the month. DATE, without the phrase YYYYMMDD, when accessed by a COBOL program, behaves as if it had been described as an unsigned elementary integer data item of usage display six digits in length, the characters positions of which, numbered from left to right, are:

Character Positions	Contents
1-2	The two low-order digits of the year in the Gregorian calendar.
3-4	Two numeric characters of the month of the year in the range 01 through 12.
5-6	Two numeric characters of the day of the month in the range 01 through 31.

3. DATE, with the phrase YYYYMMDD, is composed of the data elements year in the Gregorian calendar, month of the year, and day of the month. DATE, with the phrase YYYYMMDD, when accessed by a COBOL program, behaves as if it had been described as an unsigned elementary integer data item of usage display eight digits in length, the character positions of which, numbered from left to right, are:

Character	Contents

Positions	
1-4	Four numeric characters of the year in the Gregorian calendar.
5-6	Two numeric characters of the month of the year in the range 01 through 12.
7-8	Two numeric characters of the day of the month in the range 01 through 31.

4. DAY, without the phrase YYYYDDD, is composed of the data elements: two lower-order digits of the year in the Gregorian calendar and day of the year. DAY, without the phrase YYYYDDD, when accessed by a COBOL program, behaves as if it had been described as an unsigned elementary integer data item of usage display five digits in length, the character positions of which numbered from left to right, are:

Character Positions	Contents
1-2	The two low-order digits of the year in the Gregorian calendar.
3-5	Three numeric characters of the day of the year in the range 001 through 366.

5. DAY, with the phrase YYYYDDD, is composed of the data elements year in the Gregorian calendar and day of the year. DAY, with the phrase YYYYDDD, when accessed by a COBOL program, behaves as if it had been described as an unsigned elementary integer data item of usage display seven digits in length, the character positions of which, numbered from left to right, are:

Character Positions	Contents
1-2	The two low-order digits of the year in the Gregorian calendar.
5-7	Three numeric characters of the day of the year in the range 001 through 366.

6. TIME is composed of the data elements hours, minutes, seconds, and hundredths of a second. TIME is based on the elapsed time since midnight on a 24-hour clock. If the system does not have the facility to provide fractional parts of a second the value zero is returned for those parts that are not available. TIME, when accessed by a COBOL program, behaves as if it had been described as an unsigned elementary integer data item of usage display eight digits in length, the characters positions of which, numbered from left to right, are:

Character Positions	Contents
1-2	Two numeric characters of the hours past midnight in the range 00 through 23.
3-4	Two numeric characters of the minutes past the hour in the range 00 through 59.
5-6	Two numeric characters of the seconds past the minute in the range 00 through 59.
7-8	Two numeric characters of the hundredths of a second past the second in the range 00 through 99. 00 is returned if the system on which the ACCEPT statement is executed does not have the facility to provide the fractional part of a second.

7. DAY-OF-WEEK is composed of a single data element whose content represents the day of the week. DAY-OF-WEEK, when accessed by a COBOL program, behaves as if it had been described in a COBOL program as an unsigned elementary numeric integer data item one digit in length. In DAY-OF-WEEK, the value 1 represents Monday, 2 represents Tuesday, 3 represents Wednesday, ... , 7 represents Sunday.

8. YEAR is composed of the four-digit year, e.g., “1900” or “2000”.

FORMAT 3

1. Identifiers specified in FROM or USING clauses or literals specified in FROM or VALUE clauses provide the initial values displayed for the associated screen item during execution of an ACCEPT screen statement. For elementary data items that have no FROM, USING, or VALUE clause, the initial value is as if a MOVE statement were executed with the screen item as the receiving field. The sending item of the MOVE statement is a figurative constant which depends on the category of the screen item as follows:

Screen item	Figurative constant
Alphabetic	Alphanumeric SPACES
Alphanumeric	Alphanumeric SPACES
Alphanumeric-edited	Alphanumeric SPACES
National	National SPACES
National-edited	National SPACES
Numeric	ZEROS
Numeric-edited	ZEROS

2. The ACCEPT screen statement causes the transfer of data from each elementary screen item that is subordinate to screen-name-1 and is specified with the TO or USING clause to the data item referenced in the TO or USING clause. For the purpose of these specifications, all such screen items are considered to be referenced by the ACCEPT screen statement.

   The transfer occurs after the terminal operator has been given the opportunity to modify the elementary screen items and the operator has pressed a terminator key or a user-defined or context-dependent function key. This transfer occurs in the following manner:

   The data is transferred as if the following statement were executed:

   MOVE screen-item TO receiving-field

   where:

   receiving-field is the data item referenced in the TO or USING clause, and screen-item is the screen item.

3. During the period while the operator is able to modify each elementary screen item, each screen item is displayed on the terminal screen in accordance with any attributes specified in its screen description entry. The display may be modified as the operator selects or deselects each screen item as being the current screen item. The display of the current screen item is modified as the operator keys data characters that are consistent with the item's PICTURE clause.

4. The LINE and COLUMN phrases give the position on the terminal display screen at which the screen record associated with screen-name-1 is to start. Column and line number positions are specified in terms of alphanumeric character positions. The position is relative to the leftmost character column in the topmost line of the display that is identified as column 1 of line 1. Each subordinate elementary screen item is located relative to the start of the containing screen record. Identifier-3 and identifier-4 are evaluated once at the start of execution of the statement.

5. If the LINE phrase is not specified, the screen record starts on line 1.

6. If the COLUMN phrase is not specified, the screen record starts in column 1.
7. If the CURSOR clause is specified in special-names, then:
    a. The initial cursor position is that represented by the value of the cursor locator at the beginning of the execution of the ACCEPT screen statement. If the cursor locator does not indicate a position within an input field, the cursor shall be positioned as if the CURSOR phrase had not been specified.
    b. The data item referenced in the CURSOR clause shall be updated during the execution of an ACCEPT screen statement and prior to the execution of any imperative statement associated with any ON EXCEPTION or NOT ON EXCEPTION clauses for that ACCEPT statement. It shall be updated to give the line and column position of the cursor when the ACCEPT terminates.
8. If the CURSOR clause is not specified in special-names, the initial cursor position during the execution of an ACCEPT screen statement is the start of the first input field declared within screen-name-1.
9. Usage of the accept-screen-options is the same as when referenced in the data division's screen section.

Format 6 (user name)

Set identifier-1 to the current user name.

Format 8 (escape key)

Set identifier-1 to the last control key used to terminate the prior accept.

Format 9 (external line number)

Set identifier-1 to the current line number of the current window. This is the line at which the next default display will occur.

Format 10 (external column number)

Set identifier-1 to the current column number of the current window. This is the column at which the next default display will occur.

Format 11 (terminal-info)

Identifier-1 is filled with the following record structure, according to the current terminal:

```
01 TERMINAL-INFO-RECORD.
   05 TERMINAL-NAME PIC X(10).
   05 HAS-REVERSE PIC X. | Y/N
   05 HAS-BLINK PIC X. | Y/N
   05 HAS-UNDERLINE PIC X. | Y/N
   05 HAS-DUAL-INTENSITY PIC X. | Y/N
   05 HAS-132-COLUMNS. | Y/N
   05 HAS-COLOR. | Y/N
   05 HAS-LINE-DRAWING PIC X. | Y/N
   05 NUM-LINES  PIC 9(3).
   05 NUM-COLUMNS PIC 9(3).
   05 HAS-LOCAL-PRINTER PIC X. | Y/N
   05 HAS-VISIBLE-ATTRIBUTES PIC X. | Y/N
   05 USABLE-SCREEN-HEIGHT PIC XX COMP-X.
   05 USABLE-SCREEN-WIDTH PIC XX COMP-X.
   05 PHYSICAL-SCREEN-HEIGHT PIC XX COMP-X.
   05 PHYSICAL-SCREEN-WIDTH PIC XX COMP-X.
```

This information is not guaranteed; information which cannot be determined will be filled in from default values.

Format 12 (system-info)

Identifier-1 is filled with the follow record structure:

```
01 SYSTEM-INFORMATION.
   05 OPSYS PIC X(10). | determined operating system name
   05 USER-ID PIC X(12). | user.name property
   05 LOCAL-HOST PIC X(12). | 127000000001 is default
   05 HAS-INDEXED-READ-PREVIOUS PIC X. | Y/N
   05 HAS-RELATIVE-READ-PREVIOUS PIC X. | Y/N
   05 CAN-CHECK-INPUT-STATUS PIC X. | Y/N
   05 IS-MULTITASKING PIC X. | Y/N
   05 RUNTIME-VERSION.
      10 MAJOR-VERSION PIC 99.
      10 MINOR-VERSION PIC 99.
      10 RELEASE-VERSION PIC 99.
   05 IS-APPLET PIC X. | Y/N
```

This information is not guaranteed; information which cannot be determined will be filled in from default values.

Format 14 (standard object)

Set handle-1 to the predefined handle from the following table:

Standard-object	Predefined Handle
FIXED	FIXED-FONT
TRADITIONAL	TRADITIONAL-FONT
DEFAULT	DEFAULT-FONT
LARGE	LARGE-FONT
MEDIUM	MEDIUM-FONT
SMALL	SMALL-FONT

Format 15 (font object)

Set handle-1 to the font handle for the given font-name, where font-name is 'font[-style][-pointsize]'. The style must be bold, italic, or bolditalic. The pointsize is the font's point size, such as 12.

The font must be a font accessible to Java; for JDK 1.3+, this includes TrueType fonts. For earlier versions, the fonts included Serif, SansSerif, Monospaced and Dialog; earlier versions could have additional fonts added through the font.properties file, but this should generally be avoided.

Format 16 (window handle)

Set handle-1 to the handle for the current window.

Format 17 (handle)

Control-key-identifier specifies the identifier to hold the control key which terminates the accept, if applicable.

Accept input from the given handle or screen section identifier, setting any USING parameters specified by the handle or screen section identifier.

Format 18 (key)

Obtain a single character from the given device, generally CONSOLE, and set identifier-1 to its value. Most devices do not support this function.

Format 19 (environment)

Set identifier-1 to the contents of the System Property env-name.

Format 20 (configuration)

Set identifier-1 to the contents of the configuration parameter conf-name.

Format 22 (free)

ACCEPT FREE is synonymous with ACCEPT in Elastic COBOL; this may change in the future to allow special editing functions. This should only be used with user interaction.

Format 23 (omitted)

ACCEPT OMITTED is the same as ACCEPT with an identifier, but the result is not saved; this eliminates the need for a dummy variable to hold results when accepting a 'press any key' style message.

Control-key-identifier specifies the identifier to hold the control key which terminates the accept, if applicable.

Format 24 (input box)

Present the user with a graphical input box if possible on the system. The title of the input box may be specified. This accepts only one line of text; it will generally allow the user to scroll through its contents. This should be used only for very small user inputs. This is the input version of the DISPLAY OUTPUT.

ACQUIRE Statement

ACQUIRE acquires a program device for a transaction file. This syntax is AS/400 compatible. (DDS)

General format

ACQUIRE acquire-name FOR file-name-1

Syntax rules

1. Acquire-name may be a literal or identifier; if identifier, the contents of identifier are used. The text representation is used as the acquire-name.
2. Acquire-name must be nonnumeric and 10 characters or less.
3. File-name-1 must be a TRANSACTION file and must be open.

General rules

1. A successful ACQUIRE makes the program-device referred to by acquire-name available for I/O.
2. If unsuccessful, the I/O status code of 9H is set and any applicable USE statement is executed.
3. See DROP to drop the program device.

ADD Statement

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

General format

Format 1 (simple):

```
ADD {identifier-1 | literal-1} … TO {identifier-2 [ROUNDED] } …
        [ON SIZE ERROR imperative-statement-1]
        [NOT ON SIZE ERROR imperative-statement-2]
[END-ADD]
```

Format 2 (giving):

```
ADD {identifier-1|literal-1} … TO {identifier-2|literal-2}
        GIVING {identifier-3 [ROUNDED]} …
        [ON SIZE ERROR imperative-statement-1]
        [NOT ON SIZE ERROR imperative-statement-2]
[END-ADD]
```

Format 3 (corresponding):

```
ADD {CORRESPONDING|CORR} identifier-4 TO identifier-5 [ROUNDED]
        [ON SIZE ERROR imperative-statement-1]
        [NOT ON SIZE ERROR imperative-statement-2]
[END-ADD]
```

Format 4 (GUI component placement):

```
ADD component-1 TO component-2 [USING object-1|literal-3]
```

Format 5 (GUI component notification):

```
ADD component-1 TO SCREEN
```

Format 6 (table):

```
ADD {TABLE | MULTIPLE} source-table TO destination-table [ROUNDED]
        [FROM INDEX source-start TO source-end]
        [DESTINATION INDEX destination-start]
        [ON SIZE ERROR statement-1]
        [NOT ON SIZE ERROR statement-2]
        [END-ADD]
```

Syntax rules

1. When native arithmetic is in effect, the composite of operands described in Arithmetic statements, is determined as follows:
   a. In format 1, by using all of the operands in the statement.
   b. In format 2, by using all of the operands in the statement excluding the data items that follows the word GIVING.
   c. In format 3, by using the two corresponding operands for each separate pair of corresponding data items.
2. Identifier-1 and identifier-2 shall reference a numeric data item.
3. Literal-1 and literal-2 shall be numeric literals.
4. Identifier-3 shall reference a numeric data item or a numeric-edited data item.
5. The words CORR and CORRESPONDING are equivalent.
6. Identifier-4 and identifier-5 shall be group data items and shall not be described with level-number 66.

7. Component-1 and component-2 shall be object references to descendents of java.awt.Component (visible GUI components).
8. Component-2 shall reference a descendent of java.awt.Container which is capable of having components added.
9. Object-1 is a constraint object appropriate for the current GUI screen layout, e.g., a GridBagConstraints object for the GridBag layout.
10. Literal-3 is a constraint String appropriate for the current GUI screen layout, e.g., “NORTH” for BorderLayout.
11. Identifiers may be primitive numeric, but the ON SIZE ERROR clause is not supported in that case.

Format 6

1. source-table and destination-table must both be tables consisting of numeric items.
2. Source-start and source-end must reference valid indices within source-table.
3. Destination-start must reference a valid index within destination-table.

General rules

1. When format 1 is used, the initial evaluation shall consist of determining the value to be added, that is literal-1 or the value of the data item referenced by identifier-1, or if more than one is specified, the sum of such operands. The sum of the initial evaluation and the value of the data item referenced by identifier-2 shall be stored as the new value of the data item referenced by identifier-2.

   When standard arithmetic is in effect, the result of the initial evaluation shall be equivalent to the result of the arithmetic expression

   (operand-11 + operand-12 + ... + operand-1n)

   where the values of operand-1 are the values of literal-1 and the data items referenced by identifier-1 in the order in which they are specified in the ADD statement. The result of the sum of the initial evaluation and the value of the data item referenced by identifier-2 shall be equivalent to the result of the arithmetic expression

   (initial-evaluation + identifier-2)

   where initial-evaluation represents the result of the initial evaluation.

2. When format 2 is used, the initial evaluation shall consist of determining the sum of the operands preceding the word GIVING, that is literal-1 or the value of the data item referenced by identifier-1, and literal-2 or the value of the data item referenced by identifier-2. This value shall be stored as the new value of each data item referenced by identifier-3.

   When standard arithmetic is in effect, the result of the initial evaluation shall be equivalent to the result of the arithmetic expression

   (operand-11 + operand-12 + ... + operand-1n + operand-2)

   where the values of operand-1 are the values of literal-1 and the data items referenced by identifier-1 in the order in which they are specified in the ADD statement and the value of operand-2 is the value of either literal-2 or the data item referenced by identifier-2 in the ADD statement.

3. When format 3 is used, data items in identifier-4 shall be added to and stored in corresponding items in identifier-4.

   When standard arithmetic is in effect, the result of the addition shall be equivalent to

   (operand-1 + operand-2)

   where the value of operand-1 is the value of the data item in identifier-4 and the value of operand-2 is the value of the corresponding data item in identifier-5.

4. When native arithmetic is in effect, the compiler shall insure that enough places are carried so as not to lose any significant digits during execution.
5. Additional rules and explanations relative to this statement are given in Scope of statements; Incompatible data; ROUNDED phrase; ON SIZE ERROR phrase and size error condition; Arithmetic statements; Overlapping operands; Multiple results in arithmetic statements; CORRESPONDING phrase.
6. Format 4 adds the component component-1 to the container component-2, optionally using object-1 or literal-3 for placement information.
7. Format 5 invokes the addNotify() method of a component, creating its native peer if necessary, readying it to be rendered to the screen.

Format 6

1. Across a range of indices, a table element in source-table is added to a table element in destination-table.
2. Source-start specifies the start of the source range, defaulting to 1.
3. Source-end specifies the final element of the source range, defaulting to the final declared element of source-table.
4. Destination-start specifies the start of the destination range, defaulting to 1.

ALLOCATE Statement

The ALLOCATE statement obtains dynamic storage and returns a data pointer addressing that storage.

General format

ALLOCATE {arithmetic-expression-1 CHARACTERS | data-name-1} [INITIALIZED]
RETURNING pointer-name-1

Syntax rules

1. Data-name-1 shall reference a record description entry in the working storage section, linkage section, or file section.
2. Data-name-1 may be qualified.
3. Data-name-1 shall not reference a record description entry that contains an OCCURS DEPENDING ON clause.
4. Pointer-name-1 shall reference a data item of category data pointer.

General rules

1. Arithmetic-expression-1 specifies a number of alphanumeric character positions of storage to be allocated. If arithmetic-expression-1 does not evaluate to an integer, the result is rounded up to the next whole number.
2. If arithmetic-expression-1 evaluates to 0, the pointer referenced by pointer-name-1 is set to the predefined address NULL.
3. If data-name-1 or type-name-1 is specified, the amount of storage to be allocated is the number of alphanumeric character positions required to hold an item described by data-name-1 or type-name-1, respectively.
4. If the specified amount of storage is available for allocation, it is obtained and the pointer referenced by pointer-name-1 is set to address that storage.
5. If the specified amount of storage is not available for allocation, the pointer referenced by pointer-name-1 is set to the predefined address NULL.
6. If the INITIALIZED phrase is specified, all character positions of the allocated storage are initialized to binary zeros.
   All storage is automatically initialized to binary zeros.
7. If the INITIALIZED phrase is not specified, the content of the allocated storage is undefined.
   All storage is automatically initialized to binary zeros.
8. The allocated storage persists until explicitly released with a FREE statement or the run unit is terminated, whichever occurs first.
   The storage is automatically freed if no further references to it by any pointer exist.

ALTER Statement

The ALTER statement modifies a predetermined sequence of operations. The ALTER statement is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

General format

ALTER {procedure-name-1 TO [PROCEED TO] procedure-name-2} …

Syntax rules

1. Procedure-name-1 is the name of a paragraph that contains a sentence consisting of a GO TO statement without the DEPENDING phrase.
2. Procedure-name-2 is the name of a paragraph or section in the Procedure Division.

General rules

Execution of the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1 so that subsequent executions of the modified GO TO statement cause transfer of control to procedure-name-2.

ASSERT Statement

The ASSERT statement asserts that a condition is always true. An ASSERT is useful for debugging programs and asserting that they behave as intended. It is enabled only when compiling with the -assert flag, so normally the ASSERT does nothing, not degrading the speed or size of the program. Liberal usage of ASSERT is encouraged so when a problem does arise, the program may be compiled with -assert to help discover if anything which should always be true in fact is false.

General format

ASSERT condition-1 [AS display-parameter-1]

Syntax rules

1. condition-1 is a condition which should always be true. If it is false upon execution when compiled with -assert, an exception is thrown to be made visible to the user.
2. Display-parameter-1 is the same as the parameter to a simple DISPLAY, and it may contain references to any identifiers mentioned in condition-1 among other items. Its message will be visible and should help in diagnosing the problem.

General rules

1. ASSERT checks condition-1.
2. If condition-1 is true, it continues silently.
3. If condition-1 is false, it prints a message containing the COBOL file and line number as well as any custom message in the AS phrase.
4. If condition-1 is false, it complains with an exception stopping the current program; the exception is usually output as well, but this is dependent upon the JVM. (This exception may be caught by a CALL ON EXCEPTION, so do note that when testing subprograms.) The exact exception thrown is not guaranteed over time; as Java supports assertions more directly, a standard Java assertion exception may be thrown in the future.
5. Checking the input and output of PROCEDURE DIVISION USING parameters may be done with special assertions in the USING clause.

Example Program

```
id division.
program-id. monthassert.

data division.
working-storage section.

01 month-number pic 99.

procedure division.
main-paragraph.

   move 1 to month-number

   perform check-month

   move 13 to month-number
```

```
    perform check-month

    goback
    .

check-month.
    assert month-number>=1 and <=12
        as "Invalid month (" month-number "), should be 1..12!"

    display "the month is " month-number upon sysout
    .
```

Example output (without -assert compiler flag):

```
the month is 01
the month is 13
```

Example output (with -assert compiler flag):

```
the month is 01
assertion failure: file='monthassert.cbl', line 24 (Invalid month (13), should be 1..12!)
java.lang.RuntimeException: assertion failure: file='monthassert.cbl', line 24 (Invalid month (13), should be 1..12!)
        at monthassert.check_month(monthassert.java:193)
        at monthassert.main_paragraph(monthassert.java:178)
        at monthassert.perform(monthassert.java:375)
        at monthassert.run(monthassert.java:276)
        at monthassert.run(monthassert.java:263)
        at monthassert._main(monthassert.java:158)
        at monthassert.main(monthassert.java:305)
        at AA.main(AA.java:8)
```

BUILD Statement

Build a graphical component using AWT components (AWT).

General format

BUILD {component-identifier-1 [WITH NO SHOW] } ...

END-BUILD

Syntax rules

component-identifier-1 must have previously been built using BUILD.

General rules

1. Build or rebuild a component-identifier-1 after having changed its properties to update the display. BUILD must be the first verb used upon a component-identifier-1.
2. BUILD with the NO SHOW option builds the component structures off-screen, making them available to be used in SHOW UPON PRINTER without actually displaying the component to the user. This is used for building printer forms.
3. Warning: Do not SHOW a Frame upon the screen, always BUILD a Frame after hiding it. This is because a Frame has system resources which are allocated and de-allocated by BUILD and HIDE.

CALL Statement

The CALL statement causes control to be transferred to a specific program within the run unit, a separate program using CALL RUN. Optionally, the program may be executed within a separate thread of execution to execute concurrently with the calling thread.

General format

Format 1 (on-overflow):

CALL
[call-convention] [[EVENT] THREAD [HANDLE IN id-thread-handle]] [call-convention]
{{{identifier-1 | literal-1}} | procedure-pointer-1}
[[EVENT] THREAD [HANDLE IN id-thread-handle]] [call-convention]
OF LIBRARY library-name}
[USING [identifier-2…]
[[BY REFERENCE identifier-2…]
[BY CONTENT identifier-3…]
[BY VALUE identifier-4…]]…
[\ identifier-4 \]…
[\\ ] …
]
[{GIVING | RETURNING} INTO identifier-5]
[ON OVERFLOW imperative-statement-1]
[END-CALL]

Format 2 (on-exception):

CALL
[call-convention] [[EVENT] THREAD [HANDLE IN id-thread-handle]] [call-convention]
{{{identifier-1 | literal-1}} | procedure-pointer-1}
[[EVENT] THREAD [HANDLE IN id-thread-handle]] [call-convention]
OF LIBRARY library-name}
[USING [identifier-2…]
[[BY REFERENCE identifier-2…]
[BY CONTENT identifier-3…]
[BY VALUE identifier-4…]]…
[\ identifier-4 \]…
[\\ ] …]
[{GIVING | RETURNING} INTO identifier-5]
[ON EXCEPTION imperative-statement-1]
[NOT ON EXCEPTION imperative-statement-2]
[END-CALL]

Format 3 (on-overflow-intrinsic):

CALL

[call-convention] [[EVENT] THREAD [HANDLE IN id-thread-handle]] [call-convention]

{INTRINSIC {identifier-1 | literal-1}

[USING [identifier-2…]

[[BY REFERENCE identifier-2…]

[BY CONTENT identifier-3…]

[BY VALUE identifier-4…]]…

[\ identifier-4 \]…

[\\ ] …

]

[{GIVING | RETURNING} INTO identifier-5]

[ON OVERFLOW imperative-statement-1]

[END-CALL]

Format 4 (on-exception-intrinsic):

CALL

[call-convention] [[EVENT] THREAD [HANDLE IN id-thread-handle]] [call-convention]

{INTRINSIC {identifier-1 | literal-1}

[USING [identifier-2…]

[[BY REFERENCE identifier-2…]

[BY CONTENT identifier-3…]

[BY VALUE identifier-4…]]…

[\ identifier-4 \]…

[\\ ] …

]

[{GIVING | RETURNING} INTO identifier-5]

[ON EXCEPTION imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-CALL]

Where call-convention is:

{WITH id-call-convention LINKAGE}

| {LINKAGE TYPE id-call-convention}

| {RUN | LINKAGE TYPE PROGRAM}

Syntax rules

1. A call-convention of LINKAGE TYPE PROGRAM or RUN specifies that the called program is an entire program, not just an entry-point. This is used for calling native programs, such as notepad.exe in Windows or 'ls' in Posix, or for calling Java programs through the main entry point.
2. Literal-1 or the contents of identifier-1 is the name of the program to call; this is the externalized program-name. Procedure-pointer-1 may be used instead to hold the reference to the program to call.

3. In Elastic COBOL, the externalized program name is formed from the PROGRAM-ID clause, not the filename. The filename should be kept in sync with the PROGRAM-ID for the best behavior from the Integrated Development Environment (IDE).
4. The INTRINSIC forms are used to access the HP 3000 MPE/iX intrinsic function access library from HP. This form functions only on the HP3000 MPE/iX platform. If the keyword INTRINSIC is not used, the custom Elastic COBOL intrinsic access library is used instead. Generally, the form without using the keyword INTRINSIC is preferred in Elastic COBOL. At a later time, the INTRINSIC keyword may be ignored and all support switched to the custom Elastic COBOL library.
5. Identifier-2 is passed by reference. (To native code, identifier-2 is passed by copy/restore.)
6. Identifier-3 is passed by content.
7. Identifier-4 is passed by value. This is generally similar to by content, but this is the form preferred by Java or C code and is the preferred method for inter-language calling.
8. The \\ word is the same as the OMITTED parameter.
9. id-thread-handle holds the handle reference to the created THREAD. It must be of usage HANDLE OF THREAD. This identifier may only be used if CALL THREAD is used.

Parameters which cannot be passed in one way are automatically forced to another pass type. Generally, some usage may not be passed by reference; these are passed by content or by value instead.

USAGE	**Calling Notes**
Identifier	identifier, by reference, content or value
Table	table, by reference, content or value
Index	index, by reference, content or value
Literal	literal is passed as a variable of the same type, by content
Literal SIZE integer	literal is passed as variable of given size in bytes, by content
NULL | NULLS	the null object reference
OMITTED	specify that the parameter is not specified
Primitive-numeric	the numeric value, by value
Primitive-Boolean	the Boolean value, by value
Id-object-reference	the object reference, by value
SELF	the current program or class, by value
SUPER	the super class, by value
CONFIGURATION literal	configuration value literal, by value
DATE OF identifier	date-format of id as java.util.Date object, by value
Pointer	pointer, by reference
Procedure-pointer	procedure-pointer, by reference
Literal OF object-reference	Java field of object, by value
Literal OF id-classname	Java field of class object, by value
FUNCTION GUI (awt-id)	Java component object represented by awt-id, by value
FUNCTION EXCEPTION	the last exception object, by value
FUNCTION EXCEPTION (literal)	the last exception object of type name, by value
Id-file	file object only for calling utilities, not COBOL, by value
Id-object-hashtable (literal)	the object keyed by literal, by value

USAGE	Calling Notes
Id-condition	passed as the Boolean truth value, by content
Id-switch	passed as the Boolean truth value, by content
Id-classname	the Object representing the Class is passed, by value
Program-id	the name of the program is passed, by content
Id-entry-name	a procedure-pointer to the entry-name is passed, by content
Id-alphabet	the text representing the alphabet is passed, by content

General rules

1. The instance of the program that executes the CALL statement is the activating runtime entity.
2. The sequence of arguments in the USING phrase of the CALL statement and the sequence of formal parameters in the USING phrase of the called program's procedure division header determine the correspondence between arguments and formal parameters. The correspondence is by position, not by name. NOTE - The first argument corresponds to the first formal parameter, the second to the second, and nth to the nth. The effect of the USING phrase on the activated runtime entity is described in Procedure division, general rules.
3. The user is responsible for insuring that the data item alignment in the activating runtime entity matches that in the called program. For Elastic COBOL calling Elastic COBOL, this is not relevant.
4. The program-name is called in the following manner:
   - The program-name passed is known as the base-program-name.
   - If the configuration parameter CODE-MAPPING is enabled, then the program-name is replaced by the contents of another configuration parameter by name program-name, if present.
   - If the configuration parameter CODE-CASE is set to 1, then the program-name is converted to lower-case. If the configuration parameter CODE-CASE is set to 2, then the program-name is converted to upper-case.
   - If the configuration parameter CODE-SUFFIX is set, then its contents are appended to program-name unless program-name includes a '.' or '$' character.
   - This program-name is used in attempting to call an AcuConnect or AS/400 program.
   - If such a program exists, the call commences. Otherwise, the search continues.
   - The program-name is normalized, converting alphabetic characters to lower-case and dashes to underscores.
   - If the program-name had already been called on the current program stack, then this is a recursive call. Otherwise, it's a normal program call.
   - If a normal program call, if the same program-name has been called in the same session prior, then the prior instance of the program is used, allowing its existing WORKING-STORAGE, etc. to be maintained. Otherwise, a new instance of the given program-name will be found if possible; the normalized name and its upper-case version are checked.

- If the calling convention is LINKAGE TYPE PROGRAM or CALL RUN, then the main entry point of a Java program is used, or a full native program is loaded and executed. If a full native program is used, the USING parameters are converted to program arguments.
- If the program refers to native code, the requested calling convention is used if possible (specific to a native platform). BY VALUE should be used for native code functions. (Native code calling is not supported on all platforms.)
- Normally, the program called is a Elastic COBOL program, and its callable entry point is used.
- After the call is done, the program is forgotten if it is an INITIAL PROGRAM.
- Any return result is returned to the calling program.

1. If the program-name is not found, then the exception or overflow condition exists. If an ON EXCEPTION or ON OVERFLOW clause exists, it is executed. If no such clause exists, the Elastic COBOL runtime will notify the user that the program was not found. This default behavior should not occur in a properly functioning program after development; it may always be avoided by coding the ON EXCEPTION behavior. Afterwards, the program flow continues with the next statement after the END-CALL unless already transferred elsewhere by code in an ON EXCEPTION/ON OVERFLOW.
2. If the program-name is found, then the NOT ON EXCEPTION clause is executed, if present. Afterwards, the program flow continues with the next statement after the END-CALL unless already transferred elsewhere by the code in a NOT ON EXCEPTION clause.

CANCEL Statement

The CANCEL statement ensures that the next time the referenced program is called it will be in its initial state.

General format

CANCEL {identifier-1|literal-1|SELF} ...

Format 1 (program):

CANCEL {{identifier-1 | literal-1} OF LIBRARY library-name-1} | procedure-pointer-1 | SELF} ...

Format 2 (programs):

CANCEL ALL

Format 3 (sort):

CANCEL SORT

Syntax rules

1. Identifier-1 shall be defined as an alphanumeric or national data item.
2. Literal-1 shall be an alphanumeric or national literal.
3. Library-name-1 is present for AS/400 compatibility. Library-name-1 is an AS/400 program object library name.

General rules

1. The program to be canceled is identified by literal-1, or the content of the data item referenced by identifier-1.
2. Subsequent to the execution of a CANCEL statement, the program referred to therein cease to have any logical relationship to the run unit in which the CANCEL statement appears. If the program referenced by a successfully executed CANCEL statement in a run unit is subsequently called in that run unit, that program is in its initial state. (See State of a program.)

   NOTE - It is neither prohibited nor required that the storage of the specified program be freed by the execution of a CANCEL statement.
3. A program referenced in a CANCEL statement shall be callable by the run time entity that contains the CANCEL statement. (See Scope of names, and CALL statement.)
4. When a CANCEL statement is executed, all programs contained within the program referenced by the CANCEL statement are also canceled. The result is the same as if a valid CANCEL statement were executed for each contained program in the reverse order in which the programs appear in the separately compiled program.
5. A logical relationship to a canceled program is established only by execution of a subsequent CALL statement referencing that program.
6. No action is taken when a CANCEL statement is executed referencing a program that has not been called in this run unit or has been called and is at present canceled. Control is transferred to the next executable statement following the explicit CANCEL statement.
7. The contents of data items in external data records described by a program are not changed when that program is canceled.
8. During execution of a CANCEL statement, an implicit CLOSE statement without any optional phrases is executed for each file in the open mode that is associated with an internal file connector in the program referenced in the CANCEL statement. Any USE procedures associated with any of these files are not executed.
9. If the program to be canceled is other than a COBOL program, the effects of the CANCEL statement are defined only for those Java classes which implement the callableProgram interface; the CANCEL verb calls the cancel() method in such classes.
10. In a multi-threaded, multi-session environment such as Elastic COBOL, CANCEL is more selective about finding the program to cancel. There is a tree of program visibility, in that sessions do not normally 'see' other sessions except for SHARED-DATA, so a CANCEL will not cancel a program in another session. Rather, a CANCEL affects only a program within the current session that has been previously called from within the session. These semantics are compatible with standard COBOL when extended to a multi-session environment. It is safe to CANCEL a program within the current session without worrying about other sessions. One SESSION cannot directly CANCEL another session; it can be indirectly done by communicating through pipes, SHARED-DATA, etc.

Format 2

CANCEL ALL is ignored. This statement is accepted with a warning for compatibility, but to CANCEL ALL in a multi-threaded, multi-session environment should not be done.

Format 3

CANCEL SORT is currently ignored by the Elastic COBOL runtime. It is present for compatibility only.

CLOSE Statement

The CLOSE statement terminates the processing of reels/units and files with rewind, lock, or removal where applicable. The CLOSE statement is also used to close a graphical screen section window.

General format

CLOSE

{ file-name-1 [{REEL|UNIT} [FOR REMOVAL] | WITH {NO REWIND|LOCK}] } …

Format 2 (GfxScreen):

CLOSE WINDOW id-handle [WITH NO DISPLAY]

Syntax rules

1. The files referenced in the CLOSE statement need not all have the same organization or access.
2. The NO REWIND, REEL, and UNIT phrases may be used only with files that are of sequential organization.
3. The words REEL and UNIT are equivalent.

General rules

1. A CLOSE statement shall be executed only if the file connector referenced by file-name-1 is in an open mode.
2. For the purpose of showing the effect of various types of CLOSE statements as applied to various storage media, all files are divided into the following categories:
   a. Non-unit. A file whose input or output medium is such that the concepts of rewind and units have no meaning.
   b. Sequential single-unit. A sequential file that is entirely contained on one unit.
   c. Sequential multi-unit. A sequential file that is contained on more than one unit.
   d. Non-sequential single/multi-unit. A file with organization other than sequential, that resides on a mass storage device.
3. The results of executing each type of CLOSE for each category of file are summarized in table 12, Relationship of categories of files and the format of the CLOSE statement.

Relationship of categories of files and the format of the CLOSE statement

CLOSE statement format	File category			
	Non-unit	Sequential single-unit	Sequential multi-unit	Non-sequential single /multi-unit
CLOSE	c	c,g	a,c,g	c
CLOSE WITH LOCK	c,e	c,e,g	a,c,e,g	c,e
CLOSE WITH NO REWIND	c,h	b,c	a,b,c	N/A
CLOSE UNIT	f	f,g	f,g	N/A
CLOSE UNIT FOR REMOVAL	f	d,f,g	d,f,g	N/A

The definitions of the symbols in table 12, Relationship of categories of files and the format of the CLOSE statement, are given below. Where the definition depends on whether the file is an input, output, or input-output file, alternate definitions are given; otherwise, a definition applies to input, output, and input-output files.

a. Effect on previous units

Input files and input-output files:

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

Output files:

All units in the file prior to the current unit are closed except those units controlled by a prior CLOSE UNIT statement.

b. No rewind of current reel

The current unit is left in its current position.

c. Close file

Input files and input-output files (sequential access mode):

Input files and input-output files (random or dynamic access mode):

output files (random, dynamic, or sequential access mode):

d. Unit removal

The current unit is rewound, when applicable, and the unit is logically removed from the run unit; however, the unit may be accessed again, in its proper order of units within the file, if a CLOSE statement without the UNIT phrase is subsequently executed for this file followed by the execution of an OPEN statement for the file.

e. Close lock

When the LOCK phrase is specified on the CLOSE statement, the same physical file shall not be opened again through this file connector during this execution of this run unit.

f. Close unit

Input files and input-output files (unit media):

1. If the current unit is the last or only unit for the file, there is no unit swap, the current volume pointer remains unchanged, and the file position indicator is set to indicate that no next unit exists.

2. If another unit exists for the file, a unit swap occurs, the current volume pointer is updated to point to the next unit existing in the file, the standard beginning unit label procedure is executed, and the file position indicator is set to one less than the number of the first record existing on the new current volume. If no data records exist for the current volume, another unit swap occurs.

Output files (unit media):

The following operations take place:

1. The standard ending unit label procedure is executed.

2. A unit swap. The current volume pointer is updated to point to the new unit.

3. The standard beginning unit label procedure is executed.

4. The next executed WRITE statement that references that file directs the next logical data record to the next unit of the file.

Input files, input-output files, and output files (non-unit media):

Execution of this statement is considered successful. The file remains in the open mode, the file position indicator is unchanged, and no action takes place except as specified in general rule 4.

g. Rewind

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

h. Optional phrases ignored

The CLOSE statement is executed as if none of the optional phrases were present.

4. The execution of the CLOSE statement causes the value of the I-O status associated with file-name-1 to be updated. (See I-O status.)

5. If an optional input file is not present, no end-of-file or unit processing is performed for the file and the file position indicator and the current volume pointer are unchanged.

6. Following the successful execution of a CLOSE statement without the UNIT phrase, the record area associated with a file-name-1 is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

7. Following the successful execution of a CLOSE statement without the UNIT phrase, the file is removed from the open mode, and the file is no longer associated with the file connector.

8. The file lock and any record locks associated with the file connector referenced by file-name-1 are released at the completion of the successful execution of the CLOSE statement.

9. If more than one file-name-1 is specified in a CLOSE statement, the result of executing this CLOSE statement is the same as if a separate CLOSE statement had been written for each file-name-1 in the same order as specified in the CLOSE statement.
10. Tape media is not directly supported.
11. id-handle must be a USAGE HANDLE OF WINDOW data item.

Format 2

1. A CLOSE WINDOW is the same as the DESTROY verb, but documents that the handle being destroyed is a window. See DESTROY for more information.

COMMIT Statement

The COMMIT statement commits a file or transaction, flushing any buffers and disallowing any possibility for rollback.

General format

Format 1 (file)

COMMIT file-name-1

Format 2 (transaction)

COMMIT TRANSACTION

General rules

Format 1

1. Committing file-name-1 flushes the buffers used by file-name-1.

Format 2

1. COMMIT TRANSACTION acts on all files owned by the session.
2. This format is currently ignored by the runtime.

COMPUTE Statement

The COMPUTE statement assigns to one or more data items the value of an arithmetic expression.

General format

Format 1 (identifier):

COMPUTE { identifier-1 [ROUNDED]} … {= | EQUAL} arithmetic-expression-1
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-COMPUTE]

Format 2 (primitive):

COMPUTE { primitive-identifier-1 [ROUNDED]} … {= | EQUAL}primitive-expression-1
[END-COMPUTE]

Syntax rules

Format 1:

1. Identifier-1 shall reference either an elementary numeric item or an elementary numeric-edited item.

Format 2:

1. primitive-identifier-1 shall reference an elementary primitive numeric item, such as a USAGE JINT or JDOUBLE.

General rules

1. An arithmetic expression consisting of a single identifier or literal provides a method of setting the value of the data item referenced by identifier-1 equal to the literal or the value of the data item referenced by the single identifier.
2. Evaluation shall consist of determining the value of the arithmetic or Boolean expression.
3. Additional rules and explanations relative to this statement are given under in Arithmetic expressions; Scope of statements; Incompatible data; ROUNDED phrase; ON SIZE ERROR phrase and size error condition; Arithmetic statements; Overlapping operands; and Multiple results in arithmetic statements.

CONTINUE Statement

The CONTINUE statement is a no-operation statement. It indicates that no executable statement is present.

General format

CONTINUE

Syntax rules

The CONTINUE statement may be used anywhere a conditional statement or an imperative-statement may be used.

General rules

The CONTINUE statement has no effect on the execution of the program.

DELETE Statement

The DELETE statement logically removes an item. This may be a record from a mass storage file, a file from the MSCS, an external object reference from the active external object repository.

Format 1 (record):

DELETE file-name-1 RECORD [LOGICAL | PHYSICAL] [as400-clause] [retry-phrase]
[INVALID KEY imperative-statement-1]
[NOT INVALID KEY imperative-statement-2]
[END-DELETE]

Format 2:

DELETE FILE {file-name-1|literal-1}
[INVALID KEY imperative-statement-1]
[NOT INVALID KEY imperative-statement-2]
[END-DELETE]

Format 3:

DELETE object-reference-1

Format 4 (JFC component):

DELETE {component-object-reference-1 | id-handle-1 | ALL}

FROM {container-object-reference-2 | id-handle-2}

Format 5 (JFC notification):

DELETE {component-object-reference-1 | id-handle-1} FROM SCREEN

Where as400-clause is:

[[FORMAT format-name] [NULL-KEY-MAP null-key-identifier]]

| [[FORMAT format-name] [NULL-KEY-MAP null-key-identifier]]

Retry-phrase is defined in RETRY phrase.

Syntax rules

Format 1:

1. The format-name must be a valid format name from an indexed file assigned to an AS/400 database file.
2. The null-key-identifier must be an alphanumeric item. The NULL-KEY-MAP may be specified only for a file with the ALWNULL attribute and device type of DATABASE.
3. Id-handle must be an identifier of type USAGE HANDLE which refers to a graphical component containing Java graphical object components.

Format 2:

The DELETE FILE statement shall be specified only for a file in the closed state.

Format 4:

1. component-object-reference must be an object reference eventually inheriting from java.awt.Component.
2. container-object-reference must be an object reference eventually inheriting from java.awt.Container.

General rules

Format 1:

1. The file connector referenced by file-name-1 shall be associated with a mass storage file and shall be open in the I-O mode at the time of the execution of this statement.
2. For a file connector in the sequential access mode, the last input-output statement executed for file-name-1 prior to the execution of the DELETE statement shall have been a successfully executed READ statement. The mass storage control system (MSCS) logically removes from the file the record that was accessed by that READ statement.

3. For an indexed file connector in random or dynamic access mode, the mass storage control system (MSCS) logically removes from the file the record identified by the content of the prime record key data item associated with file-name-1. If the file does not contain the record specified by the key, the invalid key condition exists. (See Invalid key condition.)
4. For a relative file connector in random or dynamic access mode, the mass storage control system (MSCS) logically removes from the file that record identified by the content of the relative key data item associated with file-name-1. If the file does not contain the record specified by the key, the invalid key condition exists. (See Invalid key condition.)
5. After the successful execution of a DELETE statement, the identified record has been logically removed from the file and may no longer be accessed.
6. If a record lock is associated with the record to be logically removed from the file by this DELETE statement, then:
    a. If the record lock is associated with the file connector referenced by file-name-1, the record lock is released at the completion of the successful execution of the DELETE statement.
    b. Otherwise, the I-O status is set in accordance with the rules of 9.1.11, RETRY phrase.
7. If single record locking is specified for the file connector associated with file-name-1 and there is a record lock associated with that file connector on a record other than the record to be logically removed from the file by this DELETE statement, that record lock is released at the completion of the successful execution of the DELETE statement.
8. The execution of a DELETE statement does not affect the content of the record area or the content of the data item referenced by the data-name specified in the DEPENDING ON phrase of the RECORD clause associated with file-name-1.
9. The file position indicator is not affected by the execution of a DELETE statement.
10. The execution of the DELETE statement causes the value of the I-O status associated with file-name-1 to be updated. (See I-O status.)
11. Transfer of control following the successful or unsuccessful execution of the DELETE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the DELETE statement. (See Invalid key condition.)
12. The END-DELETE phrase delimits the scope of the DELETE statement. (See Scope of statements.)
13. The FORMAT clause is currently ignored by the Elastic COBOL runtime. The clause is present for AS/400 compatibility.
14. The NULL-KEY-MAP clause is currently ignored by the Elastic COBOL runtime. The clause is present for AS/400 compatibility.

Format 2:

The file(s) referenced by file-name-1 or by the external filename literal-2 are deleted from the mass storage control system (MSCS).

Format 3:

If object-reference-1 is an externally declared object, references to it in the active external object repository are removed. This statement has no effect for objects not declared as external.

Format 4:

1. This format is for object graphics support.
2. The graphical component referenced by component-object-reference-1 or id-handle-1 is removed from the graphical container container-object-reference-2 or id-handle-2.
3. If ALL is specified, then all graphical components in container-object-reference-2 or id-handle-2 are removed.

Format 5:

1. This format is for object graphics support.
2. Remove component-object-reference-1 or id-handle-1 from the screen, including its native peer, if any. The component is notified that it is being removed, allowing it to clean up if necessary.

DESTROY Statement

The DESTROY statement visibly removes and deallocates resources for graphical screen section controls. (GfxScreen)

General format

Format 1 (GfxScreen component):

DESTROY {screen-name-1 | id-handle | handle-reference}

Format 2 (GfxScreen components):

DESTROY ALL CONTROLS

Where handle-reference is:

CONTROL

{control-at-line [CELLS | PIXELS] control-at-column [CELLS | PIXELS]}

|{control-at-column [CELLS | PIXELS] control-at-line [CELLS | PIXELS]}

|{AT control-location [CELLS | PIXELS]}

Syntax rules

Format 1:

1. control-location is an integer, or numeric identifier referencing an integer, which is 4, 6 or 8 digits long. The integer is split in two, with the lines in the first half and columns in the second half.
2. Control-at-line may be any numeric within bounds of the current window, starting at 1.
3. Control-at-column may be any numeric within bounds of the current window, starting at 1.
4. For handle-reference, the handle component at the given location is the handle component used by the verb.

5. Screen-name-1 must refer to a screen section item defining a graphical screen section component.
6. Id-handle may refer to a graphical component, including a window, or it may refer to a font handle.

General rules

Format 1:

1. This format is for graphical screen section support.
2. The handle component referred to by id-handle, handle-reference, or implicitly defined by screen-name-1 is destroyed. It is visibly removed if necessary and its resources are deallocated.

Format 2:

1. This format is for graphical screen section support.
2. Destroy all graphical screen section controls on the current window.

DISPLAY Statement

The DISPLAY statement causes data to be transferred to an appropriate hardware or software device.

The DISPLAY statement is used primarily for the display of textual or graphical data to a text or graphical console. (Screen) (GfxScreen)

General format

Format 1 (Device, Screen):

DISPLAY [AND READ]
display-parameter [display-option]…
[ON EXCEPTION imperative-statement-1]
[NOT ON EXCEPTION imperative-statement-2]
[END-DISPLAY]

Format 2 (window, GfxScreen):

DISPLAY [display-parameter] display-window [modify-properties]
[ON EXCEPTION imperative-statement-1]
[NOT ON EXCEPTION imperative-statement-2]
[END-DISPLAY]

Format 3 (handle, GfxScreen):

DISPLAY display-handle [display-title-parameter] [UPON upon-handle] [modify-properties]
[ON EXCEPTION imperative-statement-1]
[NOT ON EXCEPTION imperative-statement-2]
[END-DISPLAY]

Format 4 (toolbar, GfxScreen):

DISPLAY TOOLBAR [display-title-parameter] [UPON upon-handle] [modify-properties]
[ON EXCEPTION imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-DISPLAY]

Format 5 (window title, GfxScreen):

DISPLAY

{display-parameter UPON WINDOW [TOP | BOTTOM] [CENTERED | LEFT | RIGHT] TITLE}

| {display-parameter UPON FLOATING WINDOW [window-handle] TITLE}

[ON EXCEPTION imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-DISPLAY]

Format 6 (environment):

DISPLAY display-parameter UPON ENVIRONMENT env-name

[ON EXCEPTION imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-DISPLAY]

Format 7 (configuration):

DISPLAY display-parameter UPON CONFIGURATION conf-name

[ON EXCEPTION imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-DISPLAY]

Format 8 (simple message):

DISPLAY OUTPUT display-parameter [TITLE display-parameter]

[ON EXCEPTION imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-DISPLAY]

Format 9 (screen size):

DISPLAY SCREEN SIZE screen-columns

[ON EXCEPTION imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-DISPLAY]

Format 10 (screen lines):

DISPLAY SCREEN LINES screen-lines

[ON EXCEPTION imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-DISPLAY]

Where display-parameter is:

OMITTED | {{ALL display-text}…}

Where display-option is one of the following:

WITH [NO] ADVANCING

WITH { AUTO | AUTO-SKIP | AUTOTERMINATE | AUTOMATIC }

WITH { BACKGROUND-COLOR | BACKGROUND-COLOUR } color-name

WITH { FOREGROUND-COLOR | FOREGROUND-COLOUR } color-name

WITH [NO] { BELL | BEEP }

WITH [CONVERT | CONVERSION]

[NO] TAB
WITH BLANK [SCREEN | LINE | EOL | TO END OF LINE | EOS | TO END OF SCREEN]
WITH ERASE [SCREEN | EOL | TO END OF LINE | EOS | TO END OF SCREEN]
WITH { BLINK | BLINKING }
TRAILING-SIGN
{AT | FROM} COLUMN NUMBER [PLUS | + | -] column-number [CELLS | PIXELS]
{AT | FROM} CCOLUMN NUMBER [PLUS | + | -] ccolumn-number [CELLS | PIXELS]
{AT | FROM} LINE NUMBER [PLUS | + | -] line-number [CELLS | PIXELS]
{AT | FROM} CLINE NUMBER [PLUS | + | -] cline-number [CELLS | PIXELS]
WITH FULL
WITH {HIGHLIGHT | BRIGHT | HIGH} | {NO {LOWLIGHT | DIM | LOW}}
WITH {LOWLIGHT} | {NO HIGHLIGHT}
WITH { REQUIRED | EMPTY-CHECK }
WITH {REVERSE-VIDEO | REVERSE | REVERSED}
WITH {SECURE} | {NO ECHO} | {OFF} | {NO-ECHO}
WITH { UNDERLINE | UNDERLINED }
WITH SPACE-FILL
WITH ZERO-FILL
WITH OVERLINE
WITH LEFTLINE
WITH GRIDLINE
AT at-value
WITH PROTECTED SIZE size-value
WITH PROMPT CHARACTER prompt
WITH NO PROMPT
WITH [NO] UPDATE
WITH COLOR color
ONLY PFKEYS pfkey…
FOR for-name
OF LIBRARY library-name
WITH LOCK
[UPON | ON] { display-mnemonic-name | window-handle }

Where display-window is:

{{INITIAL | STANDARD | INDEPENDENT | FLOATING} [GRAPHICAL] WINDOW}
| {FLOATING [GRAPHICAL] WINDOW UPON handle}

Where display-handle is:

display-handle-id | display-handle-component | OBJECT handle-component-integer

Where modify-properties is listed under the MODIFY statement.

Syntax rules

1. Upon-handle must be USAGE HANDLE OF WINDOW or USAGE HANDLE OF TOOL-BAR.
2. Window-handle must be of USAGE HANDLE OF WINDOW.

3. Display-handle-id must be of USAGE HANDLE.
4. Display-handle-component must be a handle component as listed in the Appendix, such as PUSH-BUTTON.
5. Handle-component-integer must be a handle component integer number as listed in the Appendix; avoid this form, present only for compatibility.
6. Display-title-parameter is the same format as display-parameter; it sets the implicit TITLE property.
7. Display-text is an identifier of class alphanumeric or numeric, literal, object reference using its toString() method, index, primitive numeric or primitive Boolean, condition or switch (text as T or F) or pointer. ALL makes no difference in the display-text and should be avoided.
8. Size-value is an integer or content of integer identifier.
9. At-value is an integer or the contents of an integer identifier. The integer must be 2, 4, 6 or 8 digits in length, divided equally between line and column.
10. Column-number is a numeric or contents of numeric identifier; integer values are used for text. It represents the starting column number, and it may be relative to the current column number.
11. Line-number is a numeric or contents of numeric identifier; integer values are used for text. It represents the starting line number, and it may be relative to the current line number.
12. Ccolumn-number is the character-based column number; Elastic COBOL displays graphically on X Windows so this value is ignored.
13. Cline-number is the character-based line number; Elastic COBOL displays graphically on X Windows so this value is ignored.
14. Prompt is a character literal or the first character of a nonnumeric identifier, or an integer literal; NO PROMPT represents a space prompt.
15. Color is a combined background and foreground-color value, an integer or the contents of an integer identifier.
16. Pfkey is used for Wang compatibility; it is an integer or the contents of an integer identifier.
17. For-name is for AS/400 compatibility; it must be at most 10 characters long.
18. Library-name is for AS/400 compatibility; it must be at most 10 characters long.
19. Display-mnemonic name is a display device.

General rules

Format 1 (Device, Screen):

1. Data may be displayed upon a variety of display-mnemonic-name devices such as SYSOUT, SYSERR, CONSOLE, SESSIONOUT. The default display-mnemonic-name device is typically CONSOLE unless running under a special environment such as a Servlet, where it is SERVLET-OUT; the default is generally the one which makes the most sense for the environment. The default may always be overridden by explicitly specifying the device.

2. For special devices, the options aside from UPON are not used.
3. For the CONSOLE device or graphical windows, most options are available.
4. The LINE and COLUMN each start at one, the upper-left corner of the screen or window.
5. The display normally advances unless WITH NO ADVANCING is specified.
6. The textual and graphical screen section displays are unified on graphical systems. Text always displays beneath graphical components. Text is always displayed at integer column and line positions; graphical components may be displayed at fractional coordinates.
7. This is the only format that normally treats screen section entries as special entities.
8. OS/400 issues:
    a. The AS/400 DATA-AREA device may use for-name, library-name and the WITH LOCK clause.
    b. The AS/400 LOCAL-DATA device may use for-name.
    c. The for-name specifies the OS/400 data area upon which to write.
    d. The library-name specifies the OS/400 library containing the data area.
    e. WITH LOCK maintains a lock on the OS/400 data area after the DISPLAY is finished.

Format 2 (window, GfxScreen):

1. Create a window, setting its initial properties. The first window created is the main window, responsible for handling the user close request.
2. INITIAL creates the main application window.
3. INDEPENDENT creates an additional application window.
4. GRAPHICAL is the same as using CELL SIZE = LABEL FONT, making the vertical size of each character cell as high as a label rather than the font height.
5. STANDARD implies TITLE-BAR, SYSTEM MENU, AUTO-MINIMIZE and USER-COLORS, and black foreground on a white background (modified by USER-COLORS into system dependent colors).
6. FLOATING is a dependent application window, for secondary control.
7. A typical GfxScreen style GUI program should begin with DISPLAY INITIAL GRAPHICAL WINDOW.

Format 3 (handle, GfxScreen):

Create a new handle component on the current or specified upon-handle window. Also set the initial properties of the component using modify-properties. This is the only method to create the MESSAGE BOX component.

Format 4 (toolbar, GfxScreen):

Create a new toolbar component on the current or specified upon-handle window. The toolbar handle should be saved using HANDLE IN so it can be used as the upon-handle for another display; this allows the toolbar to be populated with

additional components. This is the only means of creating a graphical screen section toolbar.

Format 5 (window title, GfxScreen):

Set the current or specified window's title and title position. Title position is ignored in Elastic COBOL; the operating system sets the title position.

Format 6 (environment):

Set the System Property env-name to the value display-parameter.

Format 7 (configuration):

Set the configuration parameter conf-name to the value display-parameter.

Format 8 (simple message):

Display a simple message box with the given text.

Format 9 (screen size):

Set the default number of screen columns.

Format 10 (screen lines):

Set the default number of screen lines.

General rules

ALL FORMATS

1. The END-DISPLAY phrase delimits the scope of the DISPLAY statement. (See Scope of statements.)
2. The DISPLAY statement causes the content of each operand to be transferred to the hardware device in the order listed.
3. If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed. A figurative constant other than ALL national literals shall be the alphanumeric representation of that figurative constant.
4. 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.
5. If a hardware device is not capable of receiving data of the same size as the data item being transferred, then one of the following applies:
    a. If the size of the data item being transferred exceeds the size of the data that the hardware device is capable of receiving in a single transfer, the data beginning with the leftmost character is stored aligned to the left in the receiving hardware device, and the remaining data is then transferred according to the given rules until all the data has been transferred.
    b. If the size of the data item that the hardware device is capable of receiving exceeds the size of the data being transferred, the transferred data is stored aligned to the left in the receiving hardware device.
6. When a DISPLAY statement contains more than one operand, the size of the sending item is the sum of the sizes associated with the operands, and the values of the operands are transferred in the sequence in which the operands are encountered without modifying the positioning of the hardware device between the successive operands.

7. If the UPON phrase is not specified, CONSOLE is used by default unless overridden to use SYSOUT by use of the defsys compiler option.
8. If the WITH NO ADVANCING phrase is specified, then the positioning of the hardware device shall not be reset to the next line or changed in any other way following the display of the last operand. If the hardware device is capable of positioning to a specific character position, it will remain positioned at the character position immediately following the last character of the last operand displayed. If the hardware device is not capable of positioning to a specific character position, only the vertical position, if applicable, is affected. This may cause overprinting if the hardware device supports overprinting.
9. If the WITH NO ADVANCING phrase is not specified, then after the last operand has been transferred to the hardware device, the positioning of the hardware device shall be reset to the leftmost position of the next line of the device.
10. If vertical positioning is not applicable on the hardware device, the vertical positioning shall be ignored.

FORMAT 2

1. Column and line number positions are specified in terms of alphanumeric character positions.
2. The DISPLAY statement causes the transfer of data to each elementary screen item that is subordinate to screen-name-1 and is specified with the FROM, USING, or VALUE clause, from the data item or literal referenced in the FROM, USING, or VALUE clause. For the purpose of these specifications, all such screen items are considered to be referenced by the DISPLAY screen statement.

   The transfer takes place and each elementary screen item is displayed on the terminal display subject to any editing implied in the character string specified in the PICTURE clause of each elementary screen description entry.
3. The LINE and COLUMN phrases give the position on the terminal display screen at which the screen record associated with screen-name-1 is to start. The position is relative to the leftmost character column in the topmost line of the display that is identified as column 1 of line 1. Each subordinate elementary screen item is located relative to the start of the containing screen record. Identifier-2 and identifier-3 are evaluated once at the start of execution of the statement.
4. If the LINE phrase is not specified, the screen record starts on line 1.
5. If the COLUMN phrase is not specified, the screen record starts in column 1.
6. If the execution of the DISPLAY statement is successful, the ON EXCEPTION phrase, if specified, is ignored and control is transferred to the end of the DISPLAY statement or, if the NOT ON EXCEPTION phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement that 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 DISPLAY statement.

DIVIDE Statement

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

General format

Format 1 (into):

DIVIDE {identifier-1|literal-1} INTO {identifier-2 [ROUNDED]}…
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-DIVIDE]

Format 2 (into-giving):

DIVIDE {identifier-1|literal-1} INTO {identifier-2|literal-2}
GIVING {identifier-3 [ROUNDED] } …
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-DIVIDE]

Format 3 (by-giving):

DIVIDE {identifier-2|literal-2} BY {identifier-1|literal-1}
GIVING {identifier-3 [ROUNDED]} …
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-DIVIDE]

Format 4 (into-remainder):

DIVIDE {identifier-1|literal-1} INTO {identifier-2|literal-2}
GIVING identifier-3 [ROUNDED]
REMAINDER identifier-4
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-DIVIDE]

Format 5 (by-remainder):

DIVIDE {identifier-2|literal-2} BY {identifier-1|literal-1}
GIVING identifier-3 [ROUNDED]
REMAINDER identifier-4
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-DIVIDE]

Syntax rules

1. Each identifier shall refer to an elementary numeric item, except that any identifier associated with the GIVING or REMAINDER phrase shall refer to either an elementary numeric item or an elementary numeric-edited item.
2. Each literal shall be a numeric literal.
3. When native arithmetic is in effect, the composite of operands described in 14.8.4, Arithmetic statements, is determined by using all of the operands in the statement excluding the data item that follows the word REMAINDER.

General rules

1. When format 1 is used, the initial evaluation shall consist of determining the divisor, which is literal-1 or the value of the data item referenced by identifier-1. The dividend shall be the value of the data item referenced by identifier-2. The quotient resulting from the division of the dividend by the divisor shall be stored as the new value of the data item referenced by identifier-2.
2. When formats 2 and 3 are used, the initial evaluation shall consist of determining the divisor, which is literal-1 or the value of the data item referenced by identifier-1; determining the dividend, which is literal-2 or the value of the data item referenced by identifier-2; and forming the quotient resulting from the division of the dividend by the divisor.
3. When formats 4 and 5 are used, the initial evaluation shall consist of determining the divisor, which is literal-1 or the value of the data item referenced by identifier-1; determining the dividend, which is literal-2 or the value of the data item referenced by identifier-2; forming the quotient resulting from the division of the dividend by the divisor; and forming a subsidiary quotient which shall be signed and derived from the quotient by truncation of digits at the least significant end and shall have the same number of digits and the same decimal point location as the data item referenced by identifier-3. The quotient shall be stored as the new value of the data item referenced by identifier-3. The remainder shall then be calculated and stored as the new value of the data item referenced by identifier-4. Any subscript associated with identifier-4 shall be evaluated after the new value has been stored in the data item referenced by identifier-3.
4. When formats 4 and 5 are used, the remainder shall be the result of multiplying the subsidiary quotient and the divisor and subtracting the product from the dividend.
5. When the ON SIZE ERROR phrase is specified:
    a. If a size error condition is raised during evaluation of the quotient or on storing the quotient into the data item referenced by identifier-3, the contents of the data items referenced by identifier-3 and identifier-4 shall not be changed.
    b. If a size error condition is raised when storing the remainder into the data item referenced by identifier-4, the quotient shall be stored as the new value of the data item referenced by identifier-3 but the contents of the data item referenced by identifier-4 shall not be changed.
6. Additional rules and explanations relative to this statement are given in Scope of statements; Incompatible data; ROUNDED phrase; ON SIZE ERROR phrase and size error condition; Arithmetic statements; Overlapping operands; and Multiple results in arithmetic statements.

DROP Statement

DROP releases a program device for a transaction file. This syntax is AS/400 compatible. (DDS)

General format

DROP acquire-name FOR file-name-1

Syntax rules

1. Acquire-name may be a literal or identifier; if identifier, the contents of identifier are used. The text representation is used as the acquire-name.
2. Acquire-name must be nonnumeric and 10 characters or less.
3. File-name-1 must be a TRANSACTION file and must be open.

General rules

1. A successful DROP makes the program-device referred to by acquire-name unavailable for I/O.
2. If unsuccessful, the I/O status code of 9H is set and any applicable USE statement is executed.
3. See ACQUIRE to acquire the program device.

ENTRY Statement

Create an additional externally available entry point.

ENTRY nonnumeric-literal-1 [USING identifier-1 ...]

Syntax rules

See the syntax of PROCEDURE DIVISION USING.

General rules

1. ENTRY creates an external entry point within a program.
2. The named entry point may be CALLed by another COBOL program. Execution continues immediately after the ENTRY statement.
3. The ENTRY point is available as a callableProgram only to other Elastic COBOL programs; it is not available through native code.
4. ENTRY should be used only within a PROGRAM-ID program.

EVENT Statement

Specify an event-handling procedure for a graphical component.

Format 1 (paragraph):

EXIT [stop-status-clause]

Format 2 (program):

EXIT PROGRAM [stop-status-clause]

Format 3 (all threads):

EXIT ALL PROGRAM [stop-status-clause]

FORMAT 4

EVENT DELETE object-reference-listener-1

FROM object-reference-component-1

END-EVENT

Format 6 (goback):

EXIT PROGRAM AND CONTINUE RUN UNIT

Format 7 (in-line-perform 2):
EXIT TO TEST OF PERFORM

Format 8 (method):
EXIT METHOD

Where stop-status-clause is:

WITH {ERROR | NORMAL} STATUS

| WITH {ERROR | NORMAL} STATUS

| {GIVING | RETURNING} INTO exit-status

Syntax rules

All Formats:

1. The stop-status-clause sets the RETURN-CODE special register.
2. WITH NORMAL STATUS sets RETURN-CODE to 0.
3. WITH ERROR STATUS sets RETURN-CODE to 10.
4. Exit-status must be an integer or an identifier referencing an integer. The exit-status sets RETURN-CODE in this case.

General rules

FORMAT 1

Whenever an event occurs for component-identifier, the procedure is PERFORMed.

FORMAT 2

The event procedure for component-identifier-1 is deleted, such that events on component-identifier-1 no longer PERFORM the originally given procedure.

FORMAT 3

The specified listener is added to the object-reference-component-1 to perform the given procedure-1 for all events in the listener.

FORMAT 4

object-reference-component-1 must previously have had the listener added.

Format 6

1. EXIT PROGRAM AND CONTINUE RUN UNIT is equivalent to GOBACK.
2. This statement is present for AS/400 compatibility only; GOBACK is the preferred form.

Format 7

1. EXIT TO TEST OF PERFORM is the same as EXIT PERFORM CYCLE.
2. This statement is present for AS/400 compatibility only; EXIT PERFORM CYCLE is the preferred form.

Format 8

EXIT METHOD exits the current METHOD-ID program. This statement is equivalent to GOBACK within a METHOD-ID.

EVALUATE Statement

The EVALUATE statement describes a multi-branch, multi-join structure. It may cause multiple conditions to be evaluated. The subsequent action of the object program depends on the results of these evaluations.

General format

EVALUATE selection-subject [ALSO selection-subject] …

{ {WHEN selection-object [ALSO selection-object] … } … imperative-statement-1 }
…

[WHEN OTHER imperative-statement-2]

[END-EVALUATE]

where selection-subject is:

{ identifier-1|literal-1|arithmetic-expression-1|parital-expression-1|
condition-1|TRUE|FALSE }

where selection-object is:

{[NOT] identifier-2|[NOT] literal-2|[NOT] arithmetic-expression-2|
[NOT]range-expression|condition-2| TRUE|FALSE|ANY}

where range-expression is:

{identifier-3|literal-3|arithmetic-expression-3} {THROUGH|THRU}
{identifier-4|literal-4|arithmetic-expression-4}

Syntax rules

1. The words THROUGH and THRU are equivalent.
2. The two operands in range-expression-1 shall be of the same class and shall not be of class Boolean.
3. The number of selection objects within each set of selection objects shall be equal to the number of selection subjects.
4. The classification of some selection subjects or selection objects can change for a particular WHEN phrase as follows:
    a. If the selection subject is TRUE or FALSE and the selection object is a Boolean expression that results in one Boolean character, the selection object is treated as a Boolean condition.
    b. If the selection object is TRUE or FALSE and the selection subject is a Boolean expression that results in one Boolean character, the selection subject is a treated as Boolean condition.
    c. If the selection subject is other than TRUE or FALSE and the selection object is a Boolean expression that results in one Boolean character, the selection object is treated as a Boolean expression.
    d. If the selection object is other than TRUE or FALSE and the selection subject is a Boolean expression that results in one Boolean character, the selection subject is treated as a Boolean expression.

e. If the selection subject is a class condition, omitted argument condition, or a partial expression and the selection object is a data item of the class Boolean or numeric, the selection object is treated as an identifier.

f. If the selection object is a class condition, omitted argument condition, or a partial expression and the selection subject is a data item of the class Boolean or numeric, the selection subject is treated as an identifier.

5. Each selection object within a set of selection objects shall correspond to the selection subject having the same ordinal position within the set of selection subjects according to the following rules:

   a. Identifiers, literals, or expressions appearing within a selection object shall be valid operands for comparison to the corresponding operand in the set of selection subjects. (See Relation conditions.)

   b. Condition-2 or the words TRUE or FALSE appearing as a selection object shall correspond to condition-1 or the words TRUE or FALSE in the set of selection subjects.

   c. The selection-object ANY may correspond to any selection-subject.

   d. Partial-expression-1 shall not begin with an arithmetic operator.

      NOTE - This rule prohibits adding operands to the selection-subject, as in the example + A = B.

   e. Partial-expression-1 shall correspond to a selection-subject that is an identifier, a literal, or an arithmetic-expression. Partial-expression-1 shall be a sequence of COBOL words such that, were it preceded by the corresponding selection-subject, a conditional expression would result.

6. The permissible combinations of selection subject and selection object operands are indicated in table 13, Combination of operands in EVALUATE statement.

Combination of operands in EVALUATE statement

Selection object	**Selection subject**					
	Identifier	**Literal**	**Arithmetic expression**	**Boolean expression**	**Condition**	**TRUE or FALSE**
[NOT] identifier	Y	Y	Y	Y		
[NOT] literal	Y		Y	Y		
[NOT] arithmetic-expression	Y	Y	Y			
[NOT] range-expression	Y	Y	Y			
Condition					Y	Y
Partial-expression-1	Y	Y	Y	Y		
TRUE or FALSE					Y	Y
ANY	Y	Y	Y	Y	Y	Y
The letter 'Y' indicates a permissible combination. A space indicates an invalid combination.						

General rules

1. If a selection-object is specified by partial-expression-1, that selection-object is treated as though it were specified as condition-2, where condition-2 is the conditional expression that results from applying the syntax rules for partial-expression-1. The corresponding selection-subject is treated as though it was specified by the word TRUE.

2. The execution of the EVALUATE statement operates as if each selection-subject and selection-object were evaluated and assigned a value, a range of values, or a truth-value. These values are determined as follows:

    a. Any selection subject specified by identifier-1 and any selection object specified by identifier-2, without the NOT phase, are assigned the value and class of the data item referenced by the identifier.

    b. Any selection subject specified by literal-1 and any selection object specified by literal-2, without the NOT phrase, are assigned the value and class of the specified literal. If literal-2 is a figurative constant SPACE or ZERO, it shall be assigned the class of the corresponding selection subject.

    c. Any selection subject specified by arithmetic-expression-1 and any selection object specified by arithmetic-expression-2, without the NOT phrase, are assigned a numeric value according to the rules for evaluating an arithmetic expression. (See Arithmetic expressions.)

    d. Any selection subject specified by condition-1 and any selection object specified by condition-2, are assigned a truth-value according to the rules for evaluating conditional expressions. (See Conditional expressions.)

    e. 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.

    f. Any selection object specified by the word ANY is not further evaluated.

    g. If a selection subject is specified by range-expression-1, without the NOT phrase, the range of values includes all permissible values of the selection subject that are greater than or equal to the first operand and less than or equal to the second operand according to the rules for comparison. (See Relation conditions.)

    h. If the NOT phrase is specified for a selection object, the values assigned to that item are all permissible values of the selection subject not equal to the value, or not included in the range of values, that would have been assigned to the item had the NOT phrase not been specified.

3. The execution of the EVALUATE statement then proceeds as if the values assigned to the selection subjects and selection objects were compared to determine if any WHEN phrase satisfies the set of selection subjects. This comparison proceeds as follows:

    a. Each selection object within the set of selection objects for the first WHEN phrase is compared to the selection subject having the same ordinal position within the set of selection subjects. One of the following conditions shall be satisfied if the comparison is to be satisfied:

        1. If the items being compared are assigned a value or a range of values, the comparison is satisfied if the value, or one of the range of values, assigned to the selection object is equal to the value assigned to the selection subject according to the rules for comparison. (See Relation conditions.)

2. If the items being compared is assigned truth-values, the comparison is satisfied if the items are assigned the identical truth-value.

3. If the selection object being compared is specified by the word ANY, the comparison is always satisfied regardless of the value of the selection subject.

b. If the above comparison is satisfied for every selection object within the set of selection objects being compared, the WHEN phrase containing that set of selection objects is selected as the one satisfying the set of selection subjects.

c. If the above comparison is not satisfied for one or more selection object within the set of selection objects being compared, that set of selection objects does not satisfy the set of selection subjects.

d. This procedure is repeated for subsequent sets of selection objects, in the order of their appearance in the source program, until either a WHEN phrase satisfying the set of selection subjects is selected or until all sets of selection objects are exhausted.

4. After the comparison operation is completed, execution of the EVALUATE statement proceeds as follows:

a. If a WHEN phrase is selected, execution continues with the first imperative-statement-1 following the selected WHEN phrase.

b. If no WHEN phrase is selected and a WHEN OTHER phrase is specified, execution continues with imperative-statement-2.

c. The scope of execution of the EVALUATE statement is terminated when execution reaches the end of imperative-statement-1 of the selected WHEN phrase or the end of imperative-statement-2, or when no WHEN phrase is selected and no WHEN OTHER phrase is specified. (See Scope of statements.)

EXCLUSIVE Statement

The EXCLUSIVE statement provides exclusive access to a file by locking all records in the file. This statement must occur only on a file in the open state.

General format

Format 1 (paragraph):

EXCLUSIVE file-name-1 [CONDITIONALLY]

Syntax rules

1. File-name-1 must be a valid file reference to a sharable file.
2. File-name-1 must be in the open state.
3. The record server must be running. The exclusive/un-exclusive pair share files in the open mode rather than unopened mode, so the recordserver is used to lock all records.

4. EXCLUSIVE without the CONDITIONALLY modifier waits forever for exclusive access to the file.
5. EXCLUSIVE with the CONDITIONALLY modifier sets the IO status to 93 (file locked) if exclusive access cannot be granted to the file.

EXEC Statement

The EXEC statement is used to embed for execution other languages within COBOL.

General format

Format 1 (paragraph):

EXEC language-1.
{source-code-for-language-1}
END-EXEC.

Syntax rules

The EXEC statement's contents vary according to language.

General rules

1. EXEC JAVA is supported for embedding Java source code.
   Java source code is embedded at the same location as the EXEC is found. All Java facilities available to the generated Java source code are available to the embedded Java source code. This facility is recommended only for short, self-contained segments of code. For serious inclusion of Java capability, the callableProgram interface is recommended.
2. EXEC HTML is supported for embedding HTML source code.
   Embedded HTML is sent to the Servlet output device for Servlets, CGI output for CGI programs, and SYSOUT for other cases.
3. EXEC PAGE-HTML is supported for embedding full-page HTML source code.
   Embedded HTML is sent to the Servlet output device for Servlets, CGI output for CGI programs, and SYSOUT for other cases.
4. EXEC SQL is supported for embedding SQL source code.
   Elastic COBOL supports SQL through the use of JDBC drivers.

EXIT Statement

The EXIT statement provides a common end point for a series of procedures. The EXIT PROGRAM statement marks the logical end of a called program. The EXIT METHOD statement marks the logical end of an invoked method. The EXIT FUNCTION statement marks the logical end of the execution of a function. The EXIT PERFORM format provides a means of exiting an in-line PERFORM (with or without returning to any specified test). The EXIT PARAGRAPH and EXIT SECTION statements provide a means of exiting a structured procedure without executing any of the following statements within the procedure.

General format

Format 1 (paragraph):

EXIT

Format 2 (program):

EXIT PROGRAM

[WITH {ERROR|NORMAL} STATUS |

{GIVING|RETURNING} integer-1]

Format 3 (all threads):

EXIT ALL PROGRAM

Format 4 (in-line-perform):

EXIT PERFORM [CYCLE]

Format 5 (procedure):

EXIT PARAGRAPH

Syntax rules

FORMAT 1

The EXIT statement shall appear in a sentence by itself that shall be the only sentence in the paragraph.

FORMAT 2

1. The EXIT statement shall not be specified in a declarative procedure for which the GLOBAL phrase is specified in the associated USE statement.
2. Integer-1 shall be the return-code value.
3. An EXIT PROGRAM statement shall be specified only in a program procedure division.

FORMAT 4

The EXIT PERFORM statement shall be specified only in an in-line PERFORM statement.

General rules

FORMAT 1

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

FORMAT 2

1. If the EXIT PROGRAM statement is executed in a program that is not under the control of a calling run time entity, the EXIT PROGRAM statement is treated as if it were a CONTINUE statement. No exception condition is raised even if the RAISING phrase is specified.
2. If GUI features or threading is used, then the COBOL program is treated as if under control of the GUI for purposes of exiting, and as such the EXIT PROGRAM will return control to the prior level.
3. The execution of an EXIT PROGRAM statement causes execution to continue in the calling run time entity as specified in the rules for the CALL statement. The state of the calling run time entity is not altered and is identical to that which

existed at the time it executed the CALL statement except that the contents of data items and the contents of data files shared between the calling run time entity and the called program may have been changed. If the program in which the EXIT PROGRAM statement is specified is an initial program, an implicit CANCEL statement referencing that program is executed upon return to the calling run time entity. That implicit CANCEL statement will not raise any exception conditions in the calling run time entity.

4. If an implicit CANCEL statement is executed in the calling run time entity, the exception condition exists after the execution of that implicit CANCEL statement.

FORMAT 4

The execution of an EXIT PERFORM statement without the CYCLE phrase shall cause control to be passed to an implicit CONTINUE statement immediately following the next END-PERFORM phrase.

The execution of an EXIT PERFORM statement with the CYCLE phrase shall cause control to be passed to an implicit CONTINUE statement immediately preceding the next END-PERFORM phrase.

FORMAT 5

1. If an EXIT statement with the PARAGRAPH phrase is executed, the EXIT statement causes control to be passed to an implicit CONTINUE statement immediately following the last statement in the paragraph.

FREE Statement

The FREE statement releases dynamic storage previously obtained with an ALLOCATE statement.

General format

FREE {pointer-name-1} …

Syntax rules

Pointer-name-1 shall reference a data item of category data pointer.

General rules

1. Each pointer-name-1 shall identify the start of storage that is currently allocated by the ALLOCATE statement or shall contain the predefined address NULL.
2. If the pointer referenced by pointer-name-1 contains the predefined address NULL, no action is taken, no exception condition exists, and execution continues with the next pointer-name-1, if any.
3. If the pointer referenced by pointer-name-1 identifies allocated storage, the identified storage is released and the pointer referenced by pointer-name-1 is set to the predefined address NULL. The length of storage released is the length of storage obtained by the ALLOCATE statement.

GO TO Statement

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

General format

Format 1 (unconditional):

GO TO procedure-name-1

Format 2 (depending):

GO TO {procedure-name-1} … DEPENDING ON identifier-1

Format 3 (altered):

GO TO

Syntax rules

1. Identifier-1 shall reference a numeric elementary data item that is an integer.
2. If a GO TO statement represented by format 1 appears in a consecutive sequence of imperative statements within a sentence, it shall appear as the last statement in that sequence.

General rules

1. When a GO TO statement represented by format 1 is executed, control is transferred to procedure-name-1.
2. The destination of a GO TO may be altered by the ALTER statement. This usage is obsolete and should be avoided.
3. Format 3 must be modified by an ALTER statement prior to execution. This usage is obsolete and should be avoided.

GOBACK Statement

The GOBACK statement marks the logical end of a program.

General format

GOBACK

[WITH {ERROR|NORMAL} STATUS |

{GIVING|RETURNING} integer-1]

Syntax rules

1. The GOBACK statement shall not be specified in a declarative procedure for which the GLOBAL phrase is specified in the associated USE statement.
2. The GOBACK statement shall not be specified in a method or a function.

General rules

1. If a GOBACK statement is executed in a program that is under the control of a calling run time entity, the program operates as if executing an EXIT PROGRAM statement.

2. If a GOBACK statement is executed in a program that is not under the control of a calling run time entity, the program operates as if executing a STOP RUN statement without any optional phrases.
3. If a GOBACK statement is executed within the range of a declarative procedure whose USE statement contains the GLOBAL phrase and that USE statement is specified in the same program as the GOBACK statement, the EC-FLOW-GLOBAL-GOBACK exception condition exists and the results of the execution of the GOBACK statement are unsuccessful.
4. WITH ERROR STATUS sets return-code to 10.
5. WITH NORMAL STATUS sets return-code to 0.
6. GIVING integer-1 sets return-code to integer-1.

HIDE Statement

Hide a graphical component from view. (AWT) (JFC) (GfxScreen)

General format

HIDE awt-id-1 | jfc-id-1 | {SCREEN {gfxscreen-1 | handle-id-1}} END-HIDE

Syntax rules

1. awt-id-1 is an AWT graphics component. The component must have already been built.
2. jfc-id-1 is an object reference extending java.awt.Component. The component must have already been instantiated. This is the same as invoking "setVisible" using FALSE.
3. gfxscreen-1 is a graphical screen section component identifier. The component must have been previously displayed.
4. handle-id-1 is a HANDLE reference to a graphical screen section component. The component must have been previously displayed.

General rules

The graphical component is rendered invisible.

IF Statement

The IF statement causes a condition to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false.

General format

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

NOTE - NEXT SENTENCE is an archaic feature and its use should be avoided.

Syntax rules

1. Either statement-1 and statement-2 represent an imperative statement or a conditional statement optionally proceeded by an imperative statement. A further description of the rules governing statement-1 and statement-2 is given in Scope of statements.
2. The ELSE NEXT SENTENCE phrase may be omitted if it immediately proceeds the terminal period of the sentence.
3. If the END-IF phrase is specified, the NEXT SENTENCE phrase shall not be specified.

General rules

1. The scope of the IF statement may be terminated by any of the following:
    a. An END-IF phrase at the same level of nesting.
    b. A separator period.
    c. If nested, by an ELSE phrase associated with an IF statement at a higher level of nesting as described in 14.6.4.1, Scope of statements.
2. When an IF statement is executed, the following transfers of control occur:
    a. If the condition is true and statement-1 is specified, control is transferred to the first statement of statement-1 and execution continues according to the rules for each statement specified in statement-1. If a procedure branching or conditional statement is executed that causes an explicit transfer of control, control is explicitly transferred in accordance with the rules of that statement. Upon completion of the execution of statement-1, the ELSE phrase, if specified, is ignored and control passes to the end of the IF statement.
    b. If the condition is true and the NEXT SENTENCE phrase is specified instead of statement-1, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence. This has the effect of causing a transfer of control to an implicit CONTINUE statement immediately preceding the next separator period.
    c. If the condition is false and statement-2 is specified, statement-1 or its surrogate NEXT SENTENCE is ignored, control is transferred to the first statement of statement-2, and execution continues according to the rules for each statement specified in statement-2. If a procedure branching or conditional statement is executed that causes an explicit transfer of control, control is explicitly transferred in accordance with the rules of that statement. Upon completion of the execution of statement-2, control passes to the end of the IF statement.
    d. If the condition is false and the ELSE phrase is not specified, statement-1 is ignored and control passes to the end of the IF statement.
    e. If the condition is false and the ELSE NEXT SENTENCE phrase is specified, statement-1 is ignored and control passes to the next executable sentence. This has the effect of causing a transfer of control to an implicit CONTINUE statement immediately preceding the next separator period.

3. Statement-1 and/or statement-2 may contain an IF statement. In this case, the IF statement is said to be nested. More detailed rules on nesting are given in Scope of statements.

IF statements within IF statements are considered matched IF, ELSE, and END-IF ordered combinations, proceeding from left to right. Thus, any ELSE encountered is matched with the nearest preceding IF that either has not been already matched with an ELSE or has not been implicitly or explicitly terminated. Any END-IF encountered is matched with the nearest proceeding IF that has not been implicitly or explicitly terminated.

INITIALIZE Statement

The INITIALIZE statement provides the ability to set selected data items to specified values, e.g., numeric data to zeros or alphanumeric data to spaces.

General format

The INITIALIZE statement provides the ability to set items of a certain type to a specified value when using the REPLACING clause. When the REPLACING phrase is not specified, (1) SPACE is the implied sending item for receiving items of category alphabetic,

alphanumeric, alphanumeric-edited, DBCS, national, or national-edited and (2) ZERO is the implied sending item for receiving items of category numeric or

numeric-edited. The VALUE clause is not used to initialize these values. Instead, you may want to use the INITIAL attribute of the PROGRAM-ID specification to reinitialize WORKING-STORAGE items to their VALUEs upon re-entry of the subprogram.

General format

INITIALIZE {identifier-1 [(subscripts)] }…
[WITH FILLER]

[category-name TO VALUE] [REPLACING replacing-phrase…]

Where category-name is:

{ALL |
ALPHABETIC |
ALPHANUMERIC |
NUMERIC |
ALPHANUMERIC-EDITED |
NUMERIC-EDITED |
NATIONAL |
NATIONAL-EDITED }

Where replacing-phrase is:

REPLACING category-name BY { identifier-2 | literal-1 }

Syntax rules

1. Literal-1 and the data item references by identifier-2 represent the sending area; the data item referenced by identifier-1 represents the receiving area.
2. Each category stated in the REPLACING phrase shall be a permissible category as a receiving operand in a MOVE statement where the corresponding data item referenced by identifier-2 or literal-1 is used as the sending operand.
3. The same category cannot be repeated in a REPLACING phrase.
4. The description of the data item referenced by identifier-1 or any item subordinate to identifier-1 may not contain the DEPENDING phrase of the OCCURS clause.
5. An index data item, pointer, or object may not appear as an operand of an INITIALIZE statement.
6. The data description entry for the data item referenced by identifier-1 must not contain a RENAMES clause.
7. FILLER items are skipped unless WITH FILLER is specified.

General rules

1. The key word following the word REPLACING corresponds to a category of data as defined elsewhere in this document.
2. Where identifier-1 references an elementary item or a group item, all operations are performed as if a series of MOVE statements had been written, each of which has an elementary item as its receiving field, subject to the following rules: If the REPLACING phrase is specified:
    a. 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.
    b. If identifier-1 references an elementary item, that item is initialized only if it belongs to the category specified in the REPLACING phrase.

        This initialization takes place as follows: The data item referenced by identifier-2 or literal-1 acts as the sending operand in an implicit MOVE statement to the identified item.

        All such elementary receiving fields, including all occurrences of table items within the group, are affected; the only exceptions are those fields specified in general rules 3 and 4.
3. Index data items, pointers, objects, and elementary FILLER data items are not affected by the execution of an INITIALIZE statement.
4. Any item that is subordinate to a receiving area identifier and which contains the REDEFINES clause or any item that is subordinate to such an item is excluded from this operation. However, a receiving area identifier may itself have a REDEFINES clause or be subordinate to a data item with a REDEFINES clause.
5. When the statement is written without the REPLACING phrase, data items of the categories alphabetic, alphanumeric, and alphanumeric edited are set to spaces; data items of the categories numeric and numeric edited are set to zeros. In this

case, the operation is as if each affected data item is the receiving area in an elementary MOVE statement with the indicated source literal (i.e., spaces or zeros).

6. In all cases, the content of the data item referenced by identifier-1 is set to the indicated value in the order (left to right) of the appearance of identifier-1 in the INITIALIZE statement. Within this sequence, where identifier-1 references a group item, affected elementary items are initialized in the sequence of their definition within the group.
7. If identifier-1 occupies the same storage area as identifier-2, the result of the execution of this statement is undefined, even if they are defined by the same data description entry.

INQUIRE Statement

INQUIRE retrieve the value of properties in a handle component. Many properties are in common to all handle components, but some are particular only certain types of components; these are listed in the Appendix.

The handle component must already exist; it must be in either the screen section or have been displayed and its handle saved. (GfxScreen)

General format

INQUIRE

modify-handle {PROPERTY modify-property}…

Where modify-handle is:

[WINDOW] [{id-handle | handle-reference} [(index-1 index-2)]]

Where handle-reference is:

CONTROL
{control-at-line [CELLS | PIXELS] control-at-column [CELLS | PIXELS]}
|{control-at-column [CELLS | PIXELS] control-at-line [CELLS | PIXELS]}
|{AT control-location [CELLS | PIXELS]}

Where modify-property is one of the following:

COLOR [= | IN] color-number
FOREGROUND-COLOR [= | IN] foreground-color-number
BACKGROUND-COLOR [= | IN] background-color-number
TITLE [= | IN] title
VALUE [= | IN] value
VALUE [= | IN] {MULTIPLE | TABLE} value-table
KEY [= | IN] key
{ID | IDENTIFICATION} [= | IN] id
CLASS [= | IN] class-type
SYSTEM HANDLE [= | IN] system-handle
POP-UP [MENU | AREA] [= | IN] {popup-handle | NULL | NULLS}
HANDLE [= | IN] handle
SYSTEM MENU

CONTROL FONT [= | IN] handle-font
CONTROL VALUE [= | IN] control-value
FONT [= | IN] font
Property-name [= | IN] prop-giving [LENGTH [= | IN] property-length]

Syntax rules

1. Control-at-line and control-at-column are numeric literals or the contents of numeric identifiers; control-location is an integer literal or contents of integer identifier.
2. Cline-number, ccolumn-number, clines, and csize are integer literals or the contents of integer identifiers.
3. Color-number is an integer or contents of an integer identifier, a combined foreground and background color.
4. Foreground-color is an integer or contents of an integer identifier, a number from 0 to 15.
5. Background-color is an integer or contents of an integer identifier, a number from 0 to 15.
6. Entry-1 and entry-2 are paragraph or section names; entry-1 must precede entry-2 in the source.
7. Title is a nonnumeric literal or contents of nonnumeric identifier.
8. Length is an integer literal or contents of an integer identifier.
9. Value is a nonnumeric literal or contents of a nonnumeric identifier.
10. Value-table is a table of values.
11. Key is a nonnumeric literal or contents of nonnumeric identifier.
12. Id is an integer literal or contents of an integer identifier.
13. Class-type is an integer literal or contents of an integer identifier.
14. System-handle is an integer literal or contents of an integer identifier.
15. Popup-handle is an integer literal, contents of integer identifier, or handle of component.
16. Handle is a USAGE HANDLE identifier in which the handle for the component is saved for future reference.
17. Handle-font is a USAGE HANDLE OF FONT.
18. Control-value is an integer literal or contents of integer identifier.
19. Font is a USAGE HANDLE OF FONT.
20. Property-name is a valid property name from the list of property names given for the modify-handle type.
21. Property-number is a valid property number relevant for the modify-handle type.
22. Property-length is a length of the property to use.
23. Property-value is according to the type of property-name or property-number.

24. Property-value may be set repeatedly for the same property-name by enclosing a list of property-values in parenthesis; this is useful for properties which have a cumulative effect. If the property does not have a cumulative effect, effectively only the final property-value is used.
25. Style-name is a Boolean property-name.
26. Prop-giving is an alphanumeric or numeric identifier or pointer.

General rules

1. If only WINDOW is specified, the current window is modified.
2. The HANDLE IN clause allows the handle to be saved for future reference.
3. Setting a procedure to NULLS eliminates its usage.
4. Index-1 and index-2 set certain properties to index within a component. Some complex components have internal indices which affect how the other modification properties behave. Some components may use both index-1 and index-2, only index-1, or use neither. Currently, LIST-BOX uses index-1 to set the QUERY-INDEX property.
5. The description of these properties is given in the Appendix.
6. The named property's given value is saved in the specified identifier.

INSPECT Statement

The INSPECT statement provides the ability to tally or replace occurrences of single characters or groups of characters in a data item.

General format

Format 1 (tallying):

{INSPECT|TRANSFORM} identifier-1 TALLYING tallying-phrase

Format 2 (replacing):

{INSPECT|TRANSFORM} identifier-1 REPLACING replacing-phrase

Format 3 (tallying-and-replacing):

{INSPECT|TRANSFORM} identifier-1 TALLYING tallying-phrase REPLACING replacing-phrase

Format 4 (converting):

{INSPECT|TRANSFORM} identifier-1
{CONVERTING|CONVERTING FROM|CHARACTERS FROM}
{identifier-6|literal-4} TO {identifier-7|literal-5}
[after-before-phrase]

where tallying-phrase is:

{identifier-2 FOR { CHARACTERS [after-before-phrase] |
ALL {{identifier-3|literal-1} [after-before-phrase]}… |
LEADING {{identifier-3|literal-1} [after-before-phrase]}…}…}…
TRAILING {{identifier-3 | literal-1} [after-before-phrase]}… |

where after-before-phrase is:

{AFTER INITIAL {identifier-4|literal-2}|BEFORE INITIAL {identifier-4|literal-2}}

where replacing-phrase is:

{ CHARACTERS BY replacement-item [after-before-phrase] |
ALL {{identifier-3|literal-1} BY replacement-item [after-before-phrase]}… |
LEADING {{identifier-3|literal-1} BY replacement-item [after-before-phrase]}… |
TRAILING {{identifier-3 | literal-1 } BY replacement-item [after-before-phrase]}…
FIRST {{identifier-3|literal-1} BY replacement-item [after-before-phrase]}
} …

where replacement-item is:

{identifier-5|literal-3}

Syntax rules

ALL FORMATS

1. Identifier-1 shall reference either a group item or an elementary item described implicitly or explicitly as usage display or national.
2. Identifier-3, ... , identifier-n shall reference an elementary item described implicitly or explicitly as usage display or national.
3. Each literal shall be an alphanumeric or national literal. Literal-1, literal-2, literal-3, literal-4 shall not be a figurative constant that begins with the word ALL. If literal-1, literal-2, or literal-4 is a figurative constant, it refers to an implicit one-character data item. When identifier-1 is of class national, the class of the figurative constant shall be national; otherwise, the class of the figurative constant shall be alphanumeric.
4. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, identifier-7, literal-1, literal-2, literal-3, literal-4, or literal-5 references an elementary data item or literal of class national, then all shall reference a data item or literal of class national.

FORMATS 1 AND 3

Identifier-2 shall reference an elementary numeric data item.

FORMATS 2 AND 3

1. When both literal-1 and literal-3 are specified, they shall be the same size except when literal-3 is a figurative constant, in which case it is expanded or contracted to be the size of literal-1.
2. When the CHARACTERS phrase is specified, literal-3 shall be one character in length.

FORMAT 4

When both literal-4 and literal-5 are specified they shall be the same size except when literal-5 is a figurative constant.

General rules

ALL FORMATS

1. For the purpose of determining its length, identifier-1 is treated as if it were a sending data item. (See OCCURS clause).
2. Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying

and/or replacing) begins at the leftmost character position of the data item referenced by identifier-1, regardless of its class, and proceeds from left to right to the rightmost character position as described in general rules 6 through 8.

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

    a. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 references an alphabetic, alphanumeric, Boolean, or national data item, the INSPECT statement shall treat the content of each such identifier as a character-string of the category associated with that identifier.

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

    c. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 references a category national-edited data item, the data item shall be inspected as though it had been redefined as category national and the INSPECT statement has been written to reference the redefined data item.

    d. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 references a signed numeric data item, the data item is inspected as though it had been moved to an unsigned numeric data item with length equal to the length of the signed item excluding any separate sign position, and then the rules in general rule 3b had been applied. If identifier-1 is a signed numeric item, the original value of the sign is retained upon completion of the INSPECT statement.

4. In general rules 6 through 21, all references to literal-1, literal-2, literal-3, literal-4, or literal-5 apply equally to the content of the data item referenced by identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 respectively.

5. Item identification for any identifier is done only once as the first operation in the execution of the INSPECT statement.

FORMATS 1 AND 2

1. During inspection of the content of the data item referenced by identifier-1, each properly matched occurrence of literal-1 is tallied (format 1) or replaced by literal-3 (format 2).

2. The comparison operation to determine the occurrence of literal-1 to be tallied or to be replaced, occurs as follows:

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

        - If none of LEADING, FIRST or TRAILING is specified; or

- If the LEADING adjective applies to literal-1 and literal-1 is a leading occurrence as defined in general rules 12 and 15; or
- IF the TRAILING adjective applies to literal-1 and literal-1 is a trailing occurrence as defined in general rules 12 and 15; or
- If the FIRST adjective applies to literal-1 and literal-1 is the first occurrence as defined in general rule 16.

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

c. Whenever a match occurs, tallying or replacing takes place as described in general rules 12 and 15. The character position in the data item referenced by identifier-1 immediately to the right of the rightmost character position that participated in the match is now considered to be the leftmost character position of the data item referenced by identifier-1, and the comparison cycle starts again with the first literal-1.

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

e. If the CHARACTERS phrase is specified, an implied one character operand participates in the cycle described in general rules 7a through 7d above as if it had been specified by literal-1, except that no comparison to the content of the data item referenced by identifier-1 takes place. This implied character is considered always to match the leftmost character of the content of the data item referenced by identifier-1 participating in the current comparison cycle.

3. The comparison operation defined in general rule 7 is restricted by the BEFORE and AFTER phrase as follows:

   a. If neither the BEFORE nor AFTER phrase is specified, literal-1 or the implied operand of the CHARACTERS phrase participates in the comparison operation as described in general rule 7. Literal-1 or the implied operand of the CHARACTERS phrase is first eligible to participate in matching at the leftmost character position of identifier-1.

   b. If the BEFORE phrase is specified, the associated literal-1 or the implied operand of the CHARACTERS phrase participates only in those comparison cycles that involve that portion of the content of the data item referenced by identifier-1 from its leftmost character position up to, but not including, the first occurrence of literal-2 within the content of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 7 is begun. If, on any comparison cycle, literal-1 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the content of the data item referenced by identifier-1. If there is no

occurrence of literal-2 within the content of the data item referenced by identifier-1, its associated literal-1 or the implied operand of the CHARACTERS phrase participates in the comparison operation as though the BEFORE phrase had not been specified.

c. If the AFTER phrase is specified, the associated literal-1 or the implied operand of the CHARACTERS phrase participate only in those comparison cycles that involve that portion of the content of the data item referenced by identifier-1 from the character position immediately to the right of the rightmost character position of the first occurrence of literal-2 within the content of the data item referenced by identifier-1 to the rightmost character position of the data item referenced by identifier-1. This is the character position at which literal-1 or the implied operand of the CHARACTERS phrase is first eligible to participate in matching. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 7 is begun. If, on any comparison cycle, literal-1 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the content of the data item referenced by identifier-1. If there is no occurrence of literal-2 within the content of the data item referenced by identifier-1, its associated literal-1 or the implied operand of the CHARACTERS phrase is never eligible to participate in the comparison operation.

FORMAT 1

1. The ALL, LEADING and TRAILING phrases are transitive across the operands that follow them until another ALL, LEADING or TRAILING phrase is encountered.
2. Both the ALL and LEADING phrases are transitive across the operands that follow them until another ALL or LEADING phrase is encountered.
3. The content of the data item referenced by identifier-2 is not initialized by the execution of the INSPECT statement.
4. The rules for tallying are as follows:

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

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

   c. If the CHARACTERS phrase is specified, the content of the data item referenced by identifier-2 is incremented by one for each character matched, in the sense of general rule 7e, within the content of the data item referenced by identifier-1.

   d. If the TRAILING phrase is specified, the content of the data item referenced by identifier-2 is incremented by one for the first and each subsequent occurrence of literal-1 matches within the content of the data item referenced

by identifier-1, provided that the rightmost such occurrence is at the point where comparison before in the first comparison cycle in which literal-1was eligible to participate.

5. If identifier-1, identifier-3, or identifier-4 occupies the same storage area as identifier-2, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See Overlapping operands.)

FORMATS 2 AND 3

1. The size of literal-3 or the data item referenced by identifier-5 shall be equal to the size of literal-1 or the data item referenced by identifier-3. When a figurative constant is used as literal-3, the size of the figurative constant is equal to the size of literal-1 or the size of the data item referenced by identifier-3.
2. When the CHARACTERS phrase is used, the data item referenced by identifier-5 shall be one character in length.

FORMAT 2

1. The ALL, FIRST, LEADING, and TRAILING phrases are transitive across the operands that follow them until another ALL, FIRST, LEADING, or TRAILING phrase is encountered.
2. The rules for replacement are as follows:
   a. When the CHARACTERS phrase is specified, each character matched, in the sense of general rule 7e, in the content of the data item referenced by identifier-1 is replaced by literal-3.
   b. When the adjective ALL is specified, each occurrence of literal-1 matched in the content of the data item referenced by identifier-1 is replaced by literal-3.
   c. When the adjective LEADING is specified, the first and each successive contiguous occurrence of literal-1 matched in the content of the data item referenced by identifier-1 is replaced by literal-3, provided that the leftmost occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.
   d. When the adjective FIRST is specified, the leftmost occurrence of literal-1 matched within the content of the data item referenced by identifier-1 is replaced by literal-3. This rule applies to each successive specification of the FIRST phrase regardless of the value of literal-1.
   e. When the adjective TRAILING is specified, the last and each successive contiguous occurrence of literal-1 matched in the content of the data item referenced by identifier-1 is replaced by literal-3, provided that the rightmost occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.
3. If identifier-3, identifier-4, or identifier-5 occupies the same storage area as identifier-1, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See Overlapping operands.)

FORMAT 3

A format 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 had been written with one statement being a format 1 statement with TALLYING phrases identical to

those specified in the format 3 statement, and the other statement being a format 2 statement with REPLACING phrases identical to those specified in the format 3 statement. The general rules given for matching and counting apply to the format 1 statement and the general rules given for matching and replacing apply to the format 2 statement. Subscripting associated with any identifier in the format 2 statement is evaluated only once before executing the format 1 statement.

FORMAT 4

1. A format 4 INSPECT statement is interpreted and executed as though a format 2 INSPECT statement specifying the same identifier-1 had been written with a series of ALL phrases, one for each character of literal-4. The effect is as if each of these ALL phrases referenced, as literal-1, a single character of literal-4 and referenced, as literal-3, the corresponding single character of literal-5. Correspondence between the characters of literal-4 and the characters of literal-5 is by ordinal position within the data item.
2. If identifier-4, identifier-6, or identifier-7 occupies the same storage area as identifier-1, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See Overlapping operands.)
3. The size of literal-5 or the data item referenced by identifier-7 shall be equal to the size of literal-4 or the data item referenced by identifier-6. When a figurative constant is used as literal-5, the size of the figurative constant is equal to the size of literal-4 or the size of the data item referenced by identifier-6.
4. If the same character appears more than once in the data item referenced by identifier-6 or in literal-4, the first occurrence of the character is used for replacement.

INVOKE Statement

The INVOKE statement causes a method to be invoked. The method may be either a COBOL method, defined with METHOD-ID, or a Java method. This is the general object handling verb.

A method is a procedure defined within the context of an object. The usage is similar to CALL, but it must specify not only the name, but the object on which the name is defined. A method may be defined on a per instance basis or per class basis; the first parameter is either an object or object-class-name, accordingly.

BY REFERENCE, BY CONTENT and BY VALUE parameter passing styles are available. As Elastic COBOL always operates within the Java environment, the BY VALUE is generally recommended unless invoking another COBOL method and expecting a return, in which case BY REFERENCE is used.

The PROCEDURE DIVISION USING clause has been extended to allow the expected parameter style to be explicitly specified in an invoked object; BY VALUE should be used there where possible. (BY VALUE is required for certain environments such as Enterprise JavaBeans.)

While a Elastic COBOL program is always a class object and it may be invoked, the preferred method of invoking a PROGRAM-ID style program is with the CALL verb. (JFC)

General format

Format 1 (statically bound):

INVOKE {{object-classname | object-id | SELF | SUPER} [literal-1]}
[invoke-parameters] [{GIVING | RETURNING} INTO giving-identifier]
[[ON EXCEPTION imperative-statement-1]
[NOT ON EXCEPTION imperative-statement-2]
END-INVOKE]

Format 2 (dynamically bound):

INVOKE {{object-classname | object-id | SELF | SUPER} identifier-1}
[invoke-parameters] [{GIVING | RETURNING} INTO giving-identifier]
[[ON EXCEPTION imperative-statement-1]
[NOT ON EXCEPTION imperative-statement-2]
END-INVOKE]

Format 3 (context sensitive):

INVOKE {ENVIRONMENT | CONFIGURATION} env-name env-method
[invoke-parameters] [{GIVING | RETURNING} INTO giving-identifier]
[[ON EXCEPTION imperative-statement-1]
[NOT ON EXCEPTION imperative-statement-2]
END-INVOKE]

Where invoke-parameters is:

[USING [byref-id]…
[
[BY REFERENCE byref-id…] |
[BY CONTENT bycon-id…] |
[BY VALUE byval-id…]
]…
]

Where byref-id is from the following table or the base invoke parameter table:

	Java Static	Java Dynamic	Note
Identifier	Identifier	Identifier	Copy/restore.
Identifier-table	Identifier-table	Identifier-table	Copy/restore.
Index	Index	Index	Copy/restore.
Object-id	Object Reference	Object Reference	Forced to by value.
[ALL] integer	int	N/A	Forced to by content.
[ALL] numeric-literal	double	N/A	Forced to by content.
[ALL] ZEROES	int 0	N/A	Forced to by content.
[ALL] SPACES	String space character	N/A	Forced to by content.
[ALL] QUOTES	String " or '	N/A	Forced to by content.
[ALL] nonnumeric-literal	String	N/A	Forced to by content.
True	new Boolean(true)	new Boolean(true)	Forced to by value.
False	new Boolean(false)	new Boolean(false)	Forced to by value.

Where bycon-id is from the following table or the base invoke parameter table:

	Java Static	Java Dynamic	Note
Identifier	Identifier clone	Identifier clone	Copy/restore.
Identifier-table	Identifier-table clone	Identifier-table clone	Copy/restore.
Index	Index clone	Index clone	Copy/restore.

Object-id	Object Reference clone	Object Reference clone	Forced to by value
[ALL] integer	Int	N/A	Forced to by content.
[ALL] numeric-literal	double	N/A	Forced to by content.
[ALL] ZEROES	int 0	N/A	Forced to by content.
[ALL] SPACES	String space character	N/A	Forced to by content.
[ALL] QUOTES	String " or '	N/A	Forced to by content.
[ALL] nonnumeric-literal	String	N/A	Forced to by content.
True	new Boolean(true)	new Boolean(true)	Forced to by value
False	new Boolean(false)	new Boolean(false)	Forced to by value

Where byval-id is from the following table or the base invoke parameter table:

	Java Static	**Java Dynamic**	**Note**
Identifier	Identifier	Identifier	Copy/restore.
NUMERIC DISPLAY	java.math.BigDecimal	java.math.BigDecimal	
COMP-D	java.math.BigDecimal	java.math.BigDecimal	
PACKED-DECIMAL	java.math.BigDecimal	java.math.BigDecimal	
PACKED-DECIMAL-I	java.math.BigDecimal	java.math.BigDecimal	
COMP-X	byte/short/int/long (2)	Byte()/Short()/Integer()/Long (2)	
COMP-X-REV	byte/short/int/long (2)	Byte()/Short()/Integer()/Long (2)	
BINARY	byte/short/int/long (2)	Byte()/Short()/Integer()/Long (2)	
BINARY-REV	byte/short/int/long (2)	Byte()/Short()/Integer()/Long (2)	
COMP-S	Short() / short	Short()	
COMP-1	Float() / float	Float()	
COMP-1-REV	Float() / float	Float()	
COMP-1-MVS	Float() / float	Float()	
COMP-2	Double() / double	Double()	
COMP-2-REV	Double() / double	Double()	
COMP-2-MVS	Double() / double	Double()	
BOOLEAN	Boolean() / Boolean	Boolean()	
Identifier	String (3) / char	String (3) / Character	
Identifier-table	Identifier-table	Identifier-table	Copy/restore.
Index	int	Integer()	Copy/restore.
Object-id	Object Reference (1)	Object Reference	Forced to by value.
[ALL] integer	int / long	Integer() / Long()	Forced to by content.
[ALL] numeric-literal	double	Double()	Forced to by content.
[ALL] ZEROES	int 0	Integer(0)	Forced to by content.
[ALL] SPACES	String space character	. String space character	Forced to by content
[ALL] QUOTES	String " or '	String " or '	Forced to by ntent.
[ALL] nonnumeric-literal	String	String	Forced to by content.
LENGTH OF identifier	int	Integer()	
RETURN-CODE	int	Integer()	
True	new Boolean(true)	new Boolean(true)	Forced to by value.
False	new Boolean(false)	new Boolean(false)	Forced to by value.

1. Object reference will be dereferenced for certain static calls as follows:

java.lang.Boolean	Boolean
java.lang.Byte	byte
java.lang.Short	short
java.lang.Integer	int
java.lang.Long	long
java.lang.Float	float
java.lang.Double	double
java.lang.Character	char

2. Binary storage numbers are converted to the appropriate Java size in by value

Length of 1 is byte or Byte()

Length of 2 is short or Short()

Length of 3-4 is int or Integer()

Length of 5-8 is long or Long()

3. A one character (PIC X) identifier is passed as a char or Character.

Base Invoke Parameter Table

	Java Static	**Java Dynamic**
Jbyte-id	byte	byte / new Byte(value)
Jshort-id	short	short / new Short(value)
Jint-id	int	int / new Integer(value)
Jlong-id	long	long / new Long(value)
Jchar-id	char	char / new Character(value)
Jfloat-id	float	float / new Float(value)
Jdouble-id	double	double / new Double(value)
Jboolean-id	Boolean	Boolean / new Boolean(value)
id-classname	Class object	Class object
Pointer-id	internal object	internal object
Procedure-pointer-id	internal object	internal object
file-name	internal object	internal object
Condition-id	Boolean	Boolean / new Boolean(value)
Switch-id	Boolean	Boolean / new Boolean(value)
entry-name	internal object	internal object
Alphabet-id	String	String
SYSOUT	System.out	System.out
SYSIN	System.in	System.in
SYSERR	System.err	System.err

Syntax rules

1. Giving-identifier is an object-identifier or class alphanumeric or numeric.
2. Env-name is an object name referring to an object in the current context environment; currently, the only supported env-name is SQL; this returns the current SQL java.sql.Connection object used by EXEC SQL. Env-method is the method name to invoke; any java.sql.Connection method is valid.
3. If object-id is a universal object reference, USAGE OBJECT REFERENCE without an explicit type, then the invocation is forced from format 1 (static) to format 2 (dynamic).
4. When literal-1 or identifier-1 is not specified, the method is the constructor; the constructor creates a new instance of a class; this new instance should be saved into giving-identifier. If no USING arguments are specified, this is the default constructor; if USING arguments are supplied, then the matching constructor is used.
5. Literal-1 or the contents of identifier-1 is the method-name.
6. Elastic COBOL cannot currently check the type validity of all static invocations; this may lead to the Java compiler giving errors on code that Elastic COBOL accepts when there is a type mismatch. Dynamic invocations will always compile in Java; type checking is deferred until runtime.

General rules

1. The instance of the program or class that executes the INVOKE statement is the activating runtime entity.
2. Object-id is the object instance to invoke; object-classname is the static class to invoke. Object-id is for instance methods, object-classname is for static methods.
3. The sequence of arguments in the USING phrase of the INVOKE statement and the corresponding formal parameters in the USING phrase of the invoked method's procedure division header determines the correspondence between arguments and formal parameters. The correspondence is positional and not by name equivalence.
   NOTE - The first argument corresponds to the first formal parameter, the second to the second, and the nth to the nth. The effect of the USING phrase on the activated runtime entity is described in Procedure division, general rules.
4. Execution of the INVOKE statement proceeds as follows:
   a. The object to invoke and the method-name are evaluated.
   b. All USING parameters are evaluated.
   c. The runtime system attempts to locate the given method-name on the given object or class.
   d. The method specified is made available for execution and then invoked, if possible; control is passed to the invoked method in a manner consistent with the calling convention specified for the method.
   e. If the method could not find the method or there was an error in executing the method, imperative-statement-1 is executed if given.
   f. If there was no problem in executing the method, imperative-statement-2 is executed if given. If giving-identifier is specified, it is set to the result of the method. If giving-identifier is an object identifier, SET semantics are used; the result of the invocation must be assignable to the giving-identifier. If giving-identifier is of class alphanumeric or numeric, MOVE semantics are used; an object identifier when returned is converted to its textual String representation before being moved. This allows traditional COBOL variables to receive the results of an INVOKE.
   g. After the invocation is finished, and either imperative-statement-1 or imperative-statement-2 is finished, control continues with the END-INVOKE if not already transferred elsewhere by imperative-statement-1 or imperative-statement-2.

Format 3

1. Format 3 is given to grant access to context and session sensitive runtimes. Currently, only SQL is defined, allowing direct manipulation of the EXEC SQL connection object. More env-name settings may be defined in the future.
2. Format 3 is always dynamic.

LOCK Statement

LOCK allows critical sections to be created in COBOL. Other threads may be prevented from running, an object's monitor may be waited on, or a lock may be held while a statement is performed.

General format

Format 1 (perform):
LOCK identifier FOR perform-statement

Format 2 (identifier):
LOCK WAIT identifier

Format 3 (thread):
LOCK [ALL] THREAD

Syntax rules

1. Identifier may be any identifier type other than primitives; it may be an object reference.
2. Identifier must be the same identifier in both the LOCK and UNLOCK; this means that it must be either the same object reference, or capable of being referenced directly by both threads, sessions, etc. If a value moved to an identifier used by the LOCK is the same value read by the identifier used by the UNLOCK, then the identifiers are generally fully the same. An object reference is often easiest to manage for this; it may not be null. A shared-storage item may be used for this purpose between sessions. A working-storage item may be used for this purpose between threads. Generally, a local-storage item may not be used for LOCK/UNLOCK.
3. Perform-statement is any PERFORM statement as described by PERFORM.

General rules

Format 1

1. Create a lock on an identifier while perform the perform-statement. This lock is exclusive with any others trying to obtain the same lock on the same identifier. Therefore, LOCK A FOR PERFORM and LOCK B FOR PERFORM do not block one another, but two LOCK A FOR PERFORM will be performed in serial.
2. This is an efficient method of obtaining a critical section and easier to handle from COBOL.
3. There is no UNLOCK form of this verb; the UNLOCK is done implicitly when the PERFORM is done.

Format 2

1. Wait for identifier to be notified to continue. The thread will pause until another thread informs it that it should continue by an UNLOCK WAIT.
2. This is an efficient method of obtaining a critical section and easy to integrate the lock between COBOL and Java if the lock is on an object reference. (LOCK WAIT corresponds to a Java wait(), and UNLOCK WAIT corresponds to a Java notifyAll(); either COBOL or Java may initiate either the lock or unlock.)

Format 3

1. Format 3 locks are present only for compatibility and should be avoided in favor of a format 2 or 3 lock.
2. LOCK THREAD prevents other COBOL threads from running until unlocked.
3. LOCK ALL THREAD prevents all other threads from running until unlocked; use with extreme caution as this can readily create deadlocks with Java threads.

MERGE Statement

The MERGE statement combines two or more identically sequenced files on a set of specified keys, and during the process makes records available, in merged order, to an output procedure or to an output file.

General format

MERGE filename-1 {ON {ASCENDING|DESCENDING} KEY {data-name-1}…}…
[COLLATING SEQUENCE IS alphabet-name-1]
USING filename-2 {filename-3}…
{OUTPUT PROCEDURE IS
procedure-name-1 [{THROUGH|THRU} procedure-name-2] |
GIVING {filename-4} …}

Syntax rules

1. A MERGE statement may appear anywhere in the procedure division except in the declaratives portion.
2. File-name-1 shall be described in a sort-merge file description entry in the data division.
3. If the file description entry for file-name-1 describes variable-length records, the file description entry for file-name-2 or file-name-3 shall not describe records smaller than the smallest record nor larger than the largest record described for file-name-1. If the file description entry for file-name-1 describes fixed-length records, the file description entry for file-name-2 or file-name-3 shall not describe a record that is larger than the record described for file-name-1.
4. Data-name-1 is a key data-name. Key data-names are subject to the following rules:
    a. The data items identified by key data-names shall be described in records associated with file-name-1.
    b. Key data-names may be qualified.
    c. Key data items shall not be of the class Boolean or object.
    d. The data items identified by key data-names shall not be group items that contain variable-occurrence data items.
    e. If file-name-1 has more than one record description, the data items identified by key data-names need be described in only one of the record descriptions. The same character positions referenced by a key data-name in one record description entry are taken as the key in all records of the file.

f. None of the data items identified by key data-names may be described by an entry that either contains an OCCURS clause or is subordinate to an entry that contains an OCCURS clause.

g. If the file referenced by file-name-1 contains variable-length records, all the data items identified by key data-names shall be contained within the first x character positions of the record, where x equals the minimum record size specified for the file referenced by file-name-1.

5. Alphabet-name-1 shall reference an alphabet that defines an alphanumeric collating sequence.
6. File-name-2, file-name-3, and file-name-4 shall be described in a file description entry that is not for a report file and is not a sort-merge file description entry.
7. File-names shall not be repeated within the MERGE statement.
8. No pair of file-names in a MERGE statement may be specified in the same SAME AREA, SAME SORT AREA, or SAME SORT-MERGE AREA clause. The only file-names in a MERGE statement that may be specified in the same SAME RECORD AREA clause are those associated with the GIVING phrase.
9. The words THRU and THROUGH are equivalent.
10. If file-name-4 references an indexed file, the first specification of data-name-1 shall be associated with an ASCENDING phrase and the data item referenced by that data-name-1 shall occupy the same character positions in its record as the data item associated with the prime record key for that file.
11. If the GIVING phrase is specified and the file description entry for file-name-4 describes variable-length records, the file description entry for file-name-1 shall not describe records smaller than the smallest record nor larger than the largest record described for file-name-4. If the file description entry for file-name-4 describes fixed-length records, the file description entry for file-name-1 shall not describe a record that is larger than the record described for file-name-4.

General rules

1. The MERGE statement merges all records contained on the files referenced by file-name-2 and file-name-3.
2. If the file referenced by file-name-1 contains only fixed-length records, any record in the file referenced by file-name-2 or file-name-3 containing fewer character positions than that fixed-length is space filled on the right beginning with the first character position after the last character in the record when that record is released to the file referenced by file-name-1.
3. The data-names following the word KEY are listed from left to right in the MERGE statement in order of decreasing significance without regard to how they are divided into KEY phrases. The leftmost data-name is the major key, the next data-name is the next most significant key, etc.

    a. When the ASCENDING phrase is specified, the merged sequence will be from the lowest value of the contents of the data items identified by the key data-names to the highest value, according to the rules for comparison of operands in a relation condition.

b. When the DESCENDING phrase is specified, the merged sequence will be from the highest value of the contents of the data items identified by the key data-names to the lowest value, according to the rules for comparison of operands in a relation condition.

4. When, according to the rules for the comparison of operands in a relation condition, the contents of all the key data items of one data record are equal to the contents of the corresponding key data items of one or more other data records, the order of return of these records:

   a. Follows the order of the associated input files as specified in the MERGE statement.

   b. Is such that all records associated with one input file are returned prior to the return of records from another input file.

5. The alphanumeric collating sequence that applies to the comparison of key data items of class alphabetic and class alphanumeric, and the national collating sequence that applies to the comparison of key data items of class national, are separately determined at the beginning of the execution of the MERGE statement in the following order of precedence:

   a. First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in this MERGE statement. The collating sequence associated with alphabet-name-1 applies to key data items of class alphabetic and alphanumeric; the collating sequence associated with alphabet-name-2 applies to key data items of class national.

   b. Second, the collating sequences established by the most-recently executed SET statement that specified a MERGE or SORT-MERGE phrase.

   c. Third, the collating sequences established as the program collating sequences.

6. All the records in the files referenced by file-name-2 and file-name-3 are transferred to the file referenced by file-name-1. At the start of execution of the MERGE statement, the files referenced by file-name-2 and file-name-3 shall not be in the open mode. For each of the files referenced by file-name-2 and file-name-3 the execution of the MERGE statement causes the following actions to be taken:

   a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the INPUT phrase and without a SHARING phrase had been executed. The absence of the SHARING phrase means that the sharing mode is completely determined by the SHARING clause, if any, in the file control entries for the file connectors referenced by file-name-2 and file-name-3. If an output procedure is specified, this initiation is performed before control passes to the output procedure.

   b. The logical records are obtained and released to the merge operation. Each record is obtained as if a READ statement with the NEXT and the AT END phrases had been executed. If the file referenced by file-name-1 is described with variable-length records, the size of any record written to file-name-1 is the size of that record when it was read from file-name-2 or file-name-3, regardless of the content of the data item referenced by the DEPENDING ON phrase of either a RECORD IS VARYING clause or an

OCCURS clause specified in the sort-merge file description entry for file-name-1.

c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed. If an output procedure is specified, this termination is not performed until after control passes the last statement in the output procedure. For a relative file, the content of the relative key data item is undefined after the execution of the MERGE statement.

These implicit functions are performed such that any associated USE AFTER EXCEPTION/ERROR procedures are executed. The value of the data item referenced by the DEPENDING ON phrase of a RECORD IS VARYING clause specified in the file description entry for file-name-2 or file-name-3 is undefined upon completion of the MERGE statement.

7. The output procedure may consist of any procedure needed to select, modify, or copy the records that are made available one at a time by the RETURN statement in merged order from the file referenced by file-name-1. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, INVOKE, and PERFORM statements in the range of the output procedure, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the output procedure. The range of the output procedure shall not cause the execution of any MERGE, RELEASE, or the file format of the SORT statement. (See Explicit and implicit transfers of control.)
8. If an output procedure is specified, control passes to it during execution of the MERGE statement. The compiler inserts a transfer of control mechanism at the end of the last statement in the output procedure. When control passes the last statement in the output procedure, the transfer of control mechanism provides for termination of the merge, and then passes control to the next executable statement after the MERGE statement. Before entering the output procedure, the merge procedure reaches a point at which it selects the next record in merged order when requested. The RETURN statements in the output procedure are the requests for the next record.
9. During the execution of the output procedure, no statement may be executed manipulating the file referenced by or accessing the record area associated with file-name-2 or file-name-3.
10. During the execution of any USE AFTER EXCEPTION procedure implicitly invoked while executing the MERGE statement, no statement may be executed manipulating the file referenced by or accessing the record area associated with file-name-2, file-name-3, or file-name-4.
11. If the GIVING phrase is specified, all the merged records are written on the file referenced by file-name-4 as the implied output procedure for the MERGE statement. At the start of execution of the MERGE statement, the file referenced by file-name-4 shall not be in the open mode. For each of the files referenced by file-name-4, the execution of the MERGE statement causes the following actions to be taken:

a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT and SHARING WITH NO OTHER phrases had been executed.

b. The merged logical records are returned and written onto the file. Each record is written as if a WRITE statement without any optional phrases had been executed. If the file referenced by file-name-4 is described with variable-length records, the size of any record written to file-name-4 is the size of that record when it was read from file-name-1, regardless of the content of the data item referenced by the DEPENDING ON phrase of either a RECORD IS VARYING clause or an OCCURS clause specified in the file description entry for file-name-4.

   For a relative file, the relative key data item for the first record returned contains the value '1'; for the second record returned, the value '2', etc. After execution of the MERGE statement, the content of the relative key data item indicates the last record returned to the file.

c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

   These implicit functions are performed such that any associated USE AFTER EXCEPTION/ERROR procedures are executed; however, the execution of such a USE procedure shall 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 EXCEPTION procedure associated with the file connector referenced by file-name-4 is executed; if control is returned from that USE procedure or if no such USE procedure is specified, the processing of the file is terminated as in general rule 11c above.

   The value of the data item referenced by the DEPENDING ON phrase of a RECORD IS VARYING clause specified in the sort-merge file description entry for file-name-1 is undefined upon completion of the MERGE statement for which the GIVING phrase is specified.

12. If the file referenced by file-name-4 contains only fixed-length records, any record in the file referenced by file-name-1 containing fewer character positions than that fixed-length is space filled on the right with alphanumeric space characters beginning with the first character position after the last character in the record when that record is returned to the file referenced by file-name-4.

MODIFY Statement

MODIFY modifies properties of a handle component. Many properties are in common to all handle components, but some are particular only certain types of components; these are listed in the Appendix.

The handle component must already exist; it must be in either the screen section or have been displayed and its handle saved. The modification is made immediately.

The syntax for modifications is the same as that used during creation of the component in DISPLAY.

(GfxScreen)

General format

MODIFY
modify-handle {PROPERTY modify-property}…
ON EXCEPTION imperative-statement-1
NOT ON EXCEPTION imperative-statement-2
[END-MODIFY]

Where modify-handle is:

[WINDOW] [{id-handle | handle-reference} [(index-1 index-2)]]

Where handle-reference is:

CONTROL
{control-at-line [CELLS | PIXELS] control-at-column [CELLS | PIXELS]}
|{control-at-column [CELLS | PIXELS] control-at-line [CELLS | PIXELS]}
|{AT control-location [CELLS | PIXELS]}

Where modify-property is one of the following:

AT [SCREEN] LINE NUMBER line-number [CELLS | PIXELS]
AT [SCREEN] COLUMN NUMBER column-number [CELLS | PIXELS]
LINES [=] lines [CELLS | PIXELS]
SIZE [=] size [CELLS | PIXELS]
AT CLINE NUMBER cline-number [CELLS | PIXELS]
AT CCOLUMN NUMBER ccolumn-number [CELLS | PIXELS]
CLINES [=] clines [CELLS | PIXELS]
CSIZE [=] csize [CELLS | PIXELS]
COLOR [=] color-number
FOREGROUND-COLOR [=] foreground-color-number
BACKGROUND-COLOR [=] background-color-number
HIGHLIGHT
LOWLIGHT
STANDARD
BACKGROUND-HIGH
BACKGROUND-LOW
BACKGROUND-STANDARD
EVENT PROCEDURE {[entry-1 {THROUGH | THRU} entry-2] | NULL | NULLS}
STYLE style
TITLE [LENGTH [=] length] [=] title [{GIVING|RETURNING} INTO title-giving]
VALUE [=] value [LENGTH [=] length]
VALUE [LENGTH [=] length] [=] value
VALUE [=] {MULTIPLE | TABLE} value-table
KEY [=] key
{ID | IDENTIFICATION} [=] id
CLASS [=] class-type
SYSTEM HANDLE [=] system-handle
POP-UP [MENU | AREA] [=] {popup-handle | NULL | NULLS}
HANDLE [IN|=] handle
SYSTEM MENU

CONTROL FONT [=] handle-font

CONTROL VALUE [=] control-value

FONT [=] font

Property-name [LENGTH [=] property-length] [=] [() property-value[…)] [{GIVING|RETURNING} INTO prop-giving]

PROPERTY property-number [LENGTH [=] property-length] [=] [() property-value […)] [{GIVING|RETURNING} INTO prop-giving]

[NO | NOT] Style-name [{GIVING|RETURNING} INTO prop-giving]

Syntax rules

1. Line-number, column-number, lines, and size are numeric literals or the contents of numeric identifiers.
2. Control-at-line and control-at-column are numeric literals or the contents of numeric identifiers; control-location is an integer literal or contents of integer identifier.
3. Cline-number, ccolumn-number, clines, and csize are integer literals or the contents of integer identifiers.
4. Color-number is an integer or contents of an integer identifier, a combined foreground and background color.
5. Foreground-color is an integer or contents of an integer identifier, a number from 0 to 15.
6. Background-color is an integer or contents of an integer identifier, a number from 0 to 15.
7. Entry-1 and entry-2 are paragraph or section names; entry-1 must precede entry-2 in the source.
8. Title is a nonnumeric literal or contents of nonnumeric identifier.
9. Length is an integer literal or contents of an integer identifier.
10. Value is a nonnumeric literal or contents of a nonnumeric identifier.
11. Value-table is a table of values.
12. Key is a nonnumeric literal or contents of nonnumeric identifier.
13. Id is an integer literal or contents of an integer identifier.
14. Class-type is an integer literal or contents of an integer identifier.
15. System-handle is an integer literal or contents of an integer identifier.
16. Popup-handle is an integer literal, contents of integer identifier, or handle of component.
17. Handle is a USAGE HANDLE identifier in which the handle for the component is saved for future reference.
18. Handle-font is a USAGE HANDLE OF FONT.
19. Control-value is an integer literal or contents of integer identifier.
20. Font is a USAGE HANDLE OF FONT.

21. Property-name is a valid property name from the list of property names given for the modify-handle type.
22. Property-number is a valid property number relevant for the modify-handle type.
23. Property-length is a length of the property to use.
24. Property-value is according to the type of property-name or property-number.
25. Property-value may be set repeatedly for the same property-name by enclosing a list of property-values in parenthesis; this is useful for properties which have a cumulative effect. If the property does not have a cumulative effect, effectively only the final property-value is used.
26. Style-name is a Boolean property-name.
27. Prop-giving is an alphanumeric or numeric identifier or pointer.
28. NULL and NULLS are synonymous.
29. CLINE, CCOLUMN, CLINES and CSIZE are ignored by Elastic COBOL; they are for setting character positions and sizes. They are present in Elastic COBOL for compatibility, but Elastic COBOL supports the graphical screen section even under X Windows, so they are ignored in favor of the graphical versions in Elastic COBOL.

General rules

1. If only WINDOW is specified, the current window is modified.
2. The modification of the given property is attempted.
3. If the modification is unsuccessful, imperative-statement-1 is executed if given.
4. If the modification is successful, imperative-statement-2 is executed if given.
5. The HANDLE IN clause allows the handle to be saved for future reference.
6. Setting a procedure to NULLS eliminates its usage.
7. An EVENT PROCEDURE is executed as if by PERFORM whenever an event is issued from the given component.
8. Index-1 and index-2 set certain properties to index within a component. Some complex components have internal indices which affect how the other modification properties behave. Some components may use both index-1 and index-2, only index-1, or use neither. Currently, LIST-BOX uses index-1 to set the QUERY-INDEX property.
9. The description of these properties is given in the Appendix.

MOVE Statement

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

General format

Format 1 (simple):

MOVE {identifier-1|literal-1} TO {identifier-2} …

Format 2 (corresponding):

MOVE {CORRESPONDING|CORR} identifier-3 TO identifier-4

Syntax rules

1. Primitive numerics are allowed in the MOVE statement format 1 as either source or destination. They behave like normal elementary numerics within a MOVE.
2. The words CORR and CORRESPONDING are equivalent.
3. Identifier-3 and identifier-4 shall specify group data items and shall not be reference modified.

General rules

FORMAT 1

1. Literal-1 or the data item referenced by identifier-1 represents the sending item and the identifier referenced by identifier-2 represents the receiving item.
2. Literal-1 or the content of the data item referenced by identifier-1 is moved to the data item referenced by each identifier-2 in the order specified. Any length evaluation or subscripting associated with identifier-2 is evaluated immediately before the data is moved to the respective data item.

   If identifier-1 is reference modified, subscripted, or is a function-identifier, the reference modifier, subscript, or function-identifier is evaluated only once, immediately before data is moved to the first of the receiving operands.

   The length of the data item referenced by identifier-1 is evaluated only once, immediately before the data is moved to the first of the receiving operands.

   The evaluation of the length of identifier-1 or identifier-2 may be affected by the DEPENDING ON phrase of the OCCURS clause.

   The result of the statement

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

   is equivalent to:

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

   where 'temp' is an intermediate result item provided by the runtime.
3. Any move in which the receiving operand is an elementary item and the sending operand is either a literal or an elementary data item is an elementary move.

   Alphanumeric, national, and numeric literals belong to the categories alphanumeric, national, and numeric, respectively. The category of figurative constants when used in the MOVE statement depends on the category of the receiving operand as shown in table 14, Category of figurative constants used in the MOVE statement.

   NOTE - MOVE of alphanumeric figurative constants to numeric items is an archaic feature and its use should be avoided.

Category of figurative constants used in the MOVE statement

Figurative constant	Category of receiving operand	Category of figurative constant
ALL literal, where literal is: Alphanumeric National	 - -	 Alphanumeric National
ALL symbolic character, where symbolic character is: Alphanumeric National	 - -	 Alphanumeric National
HIGH-VALUE, HIGH-VALUES; LOW-VALUE, LOW-VALUES; and QUOTE, QUOTES	Alphabetic Alphanumeric Alphanumeric-edited National National-edited Numeric Numeric-edited	Alphanumeric Alphanumeric Alphanumeric National National Alphanumeric Alphanumeric
SPACE, SPACES	Alphabetic Alphanumeric Alphanumeric-edited National National-edited	Alphabetic Alphabetic Alphabetic National National
ZERO, ZEROS, ZEROES	Alphanumeric Alphanumeric-edited National National-edited Numeric Numeric-edited	Alphanumeric Alphanumeric National National Numeric Numeric
- indicates the figurative constant category does not depend on the category of the receiving operand		

The following rules apply to an elementary move:

a. The figurative constant SPACE, an alphabetic data item, an alphanumeric-edited data item, or a national-edited data item shall not be moved to a numeric or numeric-edited data item.

b. The figurative constant ZERO, a national data item, a national literal, a national-edited data item, a numeric data item, a numeric literal, or a numeric-edited data item shall not be moved to an alphabetic data item.

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

d. A non-integer numeric literal or a non-integer numeric data item shall not be moved to a national or national-edited data item.

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

4. Any necessary conversion of data from one form of internal representation to another takes place during valid elementary moves, along with any editing specified for, or de-editing implied by, the receiving data item. Any necessary conversion from alphanumeric character to national character representation shall be performed, before any alignment, as defined for the NATIONAL-OF function without the specification of an optional substitution character. The following rules apply:

   a. When an alphanumeric, alphanumeric-edited, national, or national-edited data item is a receiving operand, alignment and any necessary space filling shall take place as defined in 13.2.8, Standard alignment rules. If the sending operand is described as being signed numeric, the operational sign

is not moved; if the operational sign occupies a separate character position, that character is not moved and the size of the sending operand is considered to be one less than its actual size in terms of standard data format characters. (See SIGN clause.) If the sending operand is numeric-edited, no de-editing takes place. If the usage of the sending operand is different from that of the receiving operand, conversion of the sending operand to the internal representation of the receiving operand takes place. If the sending operand is numeric and contains the PICTURE symbol 'P', all digit positions specified with this symbol are considered to have the value zero and are counted in the size of the sending operand.

b. When a numeric or numeric-edited item is the receiving item, alignment by decimal point and any necessary zero filling takes place as defined in Standard alignment rules. When the sending operand is numeric-edited, de-editing is implied to establish the operand's unedited numeric value, which may be signed; then the unedited numeric value is moved to the receiving field.

- When a signed numeric item is the receiving item, the sign of the sending operand is placed in the receiving item. (See SIGN clause.) Conversion of the representation of the sign takes place as necessary. If the sending operand is unsigned, a positive sign is generated for the receiving item.
- When an unsigned numeric item is the receiving item, the absolute value of the sending operand is moved and no operational sign is generated for the receiving item.
- When the sending operand is described as alphanumeric or national, an implied data item is used as the sending operand. The implied data item is an unsigned integer of category numeric with the following characteristics:

  a. If the sending operand is a data item, the implied sending item is a redefinition of the sending data item. The number of digits is the number of character positions in the sending data item unless the number of character positions is greater than 18, in which case the rightmost 18 character positions are used.

  b. If the sending operand is a figurative constant, the implied sending item contains as many digits as the receiving operand and the figurative constant is replicated in this item, from left to right, as described in the rules for figurative constants. If the receiving item is not an integer, the number of digits includes both those to the right and the left of the decimal point.

  c. If the sending operand is an alphanumeric or national literal, the implied sending operand contains as many digits as there are characters in the literal and the implied data item contains the characters in the literal. If the number of characters exceeds 18, the size of the sending operand is 18 digits and only the rightmost 18 alphanumeric characters in the literal are used."

c. When a receiving item is described as alphabetic, justification and any necessary space filling takes place as defined in 13.2.8, Standard alignment rules.

5. Any move that is not an elementary move is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area will be filled without consideration for the individual elementary or group items contained within either the sending or receiving area, except as noted in the OCCURS clause.
6. Data in table 15, Validity of types of MOVE statements, summarizes the validity of the various types of MOVE statements. The general rule reference indicates the rule that prohibits the move or that describes the behavior of a valid move.

Validity of types of MOVE statements

Operand		**Category of sending**			
		Alphabetic	**Alphanumeric-** edited, Alphanumeric	**National, National-** edited	**Numeric, Numeric-** edited
Alphabetic		Yes/4c	Yes/4a	Yes/4a	No/3a
Alphanumeric		Yes/4c	Yes/4a	Yes/4a	Yes/4b
Alphanumeric-edited		Yes/4c	Yes/4a	Yes/4a	No/3a
National		No/3b	No/3c	Yes/4a	Yes/4b
National-edited		No/3b	No/3c	Yes/4a	No/3a
Numeric	Integer	No/3b	Yes/4a	Yes/4a	Yes/4b
	Non-integer	No/3b	No/3c	No/3d	Yes/4b
Numeric-edited		No/3b	Yes/4a	Yes/4a	Yes/4b

FORMAT 2

Data items within identifier-3 are selected to be moved to selected data items within identifier-4 according to the rules specified in CORRESPONDING phrase. The results are the same as if the user had referred to each pair of corresponding identifiers in separate MOVE statements.

MULTIPLY Statement

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

General format

Format 1 (by):

MULTIPLY {identifier-1|literal-1} BY {identifier-2 [ROUNDED]} …
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-MULTIPLY]

Format 2 (by-giving):

MULTIPLY {identifier-1|literal-1} BY {identifier-2|literal-2}
GIVING {identifier-3 [ROUNDED]}…
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-MULTIPLY]

Syntax rules

1. Each identifier shall refer to a numeric elementary item, except that in format 2 each identifier following the word GIVING shall refer to either an elementary numeric item or an elementary numeric-edited item.
2. Each literal shall be a numeric literal.

General rules

1. When format 1 is used and native arithmetic is in effect, the initial evaluation consists of determining the multiplier, which is literal-1 or the value of the data item referenced by identifier-1. The multiplicand is the value of the data item referenced by identifier-2. The product of the multiplier and the multiplicand is stored as the new value of the data item referenced by identifier-2.
2. When format 2 is used, the initial evaluation consists of determining the multiplier, which is literal-1 or the value of the data item referenced by identifier-1; determining the multiplicand, which is literal-2 or the value of the data item referenced by identifier-2; and forming the product of the multiplier and the multiplicand. The product is stored as the new value of each data item referenced by identifier-3.
3. Additional rules and explanations relative to this statement are given in Scope of statements; Incompatible data; ROUNDED phrase; ON SIZE ERROR phrase and size error condition; Arithmetic statements; Overlapping operands; and Multiple results in arithmetic statements.

NOTE Statement

NOTE embeds a commentary sentence. This use is archaic and present for compatibility only; it should not be used in new code. Use a standard comment instead.

General format

NOTE comments .

Syntax rules

1. Comments are ignored until the next period character.

OPEN Statement

The OPEN statement initiates the processing of files.

General format

OPEN {file-open-mode {[before-file]… file-name-1 [after-file]…}…}…

Where file-open-mode is:

[EXCLUSIVE|SHARED] INPUT
[EXCLUSIVE|SHARED] OUTPUT

[EXCLUSIVE|SHARED] I-O
[EXCLUSIVE|SHARED] EXTEND
SHARED [I-O]

Where before-file is:

SHARING WITH ALL OTHER
SHARING WITH NO OTHER
SHARING WITH READ ONLY
retry-phrase

Where after-file is:

REVERSED
WITH NO REWIND
WITH LOCK
WITH MASS-UPDATE
ALLOWING NO OTHERS
ALLOWING READERS
ALLOWING WRITERS
ALLOWING UPDATERS
ALLOWING ALL

Where retry-phrase is defined in RETRY phrase.

Syntax rules

1. The EXTEND phrase shall only be used for files in the sequential access mode for which the LINAGE clause has not been specified.
2. The files referenced in the OPEN statement need not all have the same organization or access.
3. The NO REWIND phrase may only be specified for sequential files.
4. The NO REWIND phrase shall be specified only when the INPUT or OUTPUT phrase is specified.
5. If SHARING ALL is specified and the INPUT phrase is not specified, the LOCK MODE clause shall be specified in the file control entry for file-name-1.
6. A file-open-mode of SHARED uses SHARING WITH READ ONLY.
7. A file-open-mode of EXCLUSIVE, or an after-file WITH LOCK, uses SHARING WITH READ ONLY for INPUT files, and SHARING WITH NO OTHER for other files.
8. MASS-UPDATE implies SHARING WITH NO OTHERS and enables write caching.
9. ALLOWING NO OTHERS is the same as SHARING WITH NO OTHER.
10. ALLOWING READERS is the same as SHARING WITH READ ONLY.
11. ALLOWING UPDATERS and ALLOWING ALL is the same as SHARING WITH ALL OTHER.
12. A TRANSACTION file may be opened only for I-O.

General rules

1. The successful execution of an OPEN statement determines the availability of the file and results in the file connector referenced by file-name-1 being in an

open mode. The successful execution of an OPEN statement associates the file with file-name-1 through a file connector.

The key words INPUT, OUTPUT, I-O, and EXTEND specify the type of input-output operations intended for the file through the file connector and establish the open mode. (See table 18, Permissible statements.)

A file is available if it is physically present and is recognized by the IOCS. Table 16, Opening available and unavailable files (file not currently open), shows the results of opening available and unavailable files that are not currently open. Table 17, Opening available shared files that are currently open by another file connector, shows the results of opening available and unavailable files that are currently open by another file connector.

Opening available and unavailable files (file not currently open)

Open mode	File is available	File is unavailable
INPUT	Normal open	Open is unsuccessful
INPUT (optional file)	Normal open	Normal open; the first read causes the at end condition or invalid key condition
I-O	Normal open	Open is unsuccessful
I-O (optional file)	Normal open	Open causes the file to be created
OUTPUT	Normal open; the file contains no records	Open causes the file to be created
EXTEND	Normal open	Open is unsuccessful
EXTEND (optional file)	Normal open	Open causes the file to be created

Opening available shared files that are currently open by another file connector

Most restrictive existing sharing mode and open mode		OPEN				
		sharing with no other	sharing with read only		sharing with all other	
Open request		extend I-O input output	extend I-O output	input	extend I-O output	input
SHARING WITH NO OTHER	EXTEND I-O INPUT OUTPUT	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open
SHARING WITH READ ONLY	EXTEND I-O	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open	Normal open
	INPUT	Unsuccessful open	Unsuccessful open	Normal open	Unsuccessful open	Normal open
	OUTPUT	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open
SHARING WITH ALL OTHER	EXTEND I-O	Unsuccessful Open	Unsuccessful open	Unsuccessful open	Normal open	Normal open
	INPUT	Unsuccessful open	Normal open	Normal open	Normal open	Normal open
	OUTPUT	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open	Unsuccessful open

2. The successful execution of an OPEN statement makes the associated record area available to the program. If the file connector associated with file-name is an external file connector, there is only one record area associated with the file connector for the run unit.

3. When a file is not in an open mode, no statement may be executed that references the file, either explicitly or implicitly, except for a MERGE statement with the USING or GIVING phrase, an OPEN statement, or a SORT statement with the USING or GIVING phrase.
4. The OPEN statement for a report file shall be executed prior to the execution of an INITIATE statement for any reports contained in the file.
5. An OPEN statement shall be successfully executed prior to the execution of any of the permissible input-output statements. In table 18, Permissible statements, 'X' at an intersection indicates that the specified statement, used in the access mode given for that row, may be used with the open mode given at the top of the column.

Permissible statements

File access mode	Open mode				
	Statement	**Input**	**Output**	**I-O**	**Extend**
Sequential	READ	X		X	
	WRITE		X		X
	REWRITE			X	
Sequential (Relative and indexed files only)	START	X		X	
	DELETE			X	
Random	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START				
	DELETE			X	
Dynamic	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START	X		X	
	DELETE			X	

6. A file may be opened with the INPUT, OUTPUT, EXTEND, and I-O phrases in the same run unit. Following the initial execution of an OPEN statement for a file through a file connector, each subsequent OPEN statement executed for that file through that same file connector shall be preceded by the execution of a CLOSE statement, without the REEL, UNIT, or LOCK phrase, for that file.
7. Execution of the OPEN statement does not obtain or release the first data record.
8. If during the execution of an OPEN statement a file attribute conflict condition occurs, the execution of the OPEN statement is unsuccessful. The validation of fixed-file attributes may vary depending on the organization and/or storage medium of the file. (See File attributes.)
9. The NO REWIND phrase shall only be used with sequential single reel/unit files. (See CLOSE statement.)
10. The NO REWIND phrase will be ignored if it does not apply to the storage medium on which the file resides.
11. If the storage medium for the file permits rewinding, the following rules apply:

a. When neither the EXTEND, nor the NO REWIND phrase is specified, execution of the OPEN statement causes the file to be positioned at its beginning.

b. When the NO REWIND phrase is specified, execution of the OPEN statement does not cause the file to be repositioned; the file shall be already positioned at its beginning prior to execution of the OPEN statement.

12. If a file opened with the INPUT phrase is an optional file that is not present, the OPEN statement sets the file position indicator to indicate that an optional input file is not present.

13. For sequential and relative files, when files are opened with the INPUT or I-O phrase, the file position indicator is set to 1. For indexed files, when files are opened with the INPUT or I-O phrase, the file position indicator is set to the characters that have the lowest ordinal position in the collating sequence associated with the file, and the prime record key is established as the key of reference.

14. When the EXTEND phrase is specified, the OPEN statement positions the file immediately after the last logical record for that file. The last logical record for a sequential file is the last record written in the file. The last logical record for a relative file is the currently existing record with the highest relative record number. The last logical record for an indexed file is the currently existing record with the highest prime key value.

15. The OPEN statement with the I-O phrase shall reference a file that supports the input and output operations that are permitted for the organization of that file when opened in the I-O mode. The execution of the OPEN statement with the I-O phrase places the referenced file in the open mode for both input and output operations.

16. For an optional file that is unavailable, the successful execution of an OPEN statement with an EXTEND or I-O phrase creates the file. This creation takes place as if the following statements were executed in the order shown:

    OPEN OUTPUT file-name.

    CLOSE file-name.

    These statements are followed by execution of the OPEN statement specified in the source program.

17. The successful execution of an OPEN statement with the OUTPUT phrase creates the file. After the successful creation of a file, that file contains no data records. If physical pages have meaning for the file, the positioning of the output medium with respect to physical page boundaries is performed following the successful execution of the OPEN statement, whether or not the LINAGE clause is specified for the file.

18. Upon successful execution of the OPEN statement, the current volume pointer is set:

    a. To point to the first or only reel/unit for an available input or I-O file.

    b. To point to the reel/unit containing the last logical record for an extend file.

    c. To point to the new reel/unit for an unavailable output, I-O, or extend file.

19. The execution of the OPEN statement causes the value of the I-O status associated with file-name to be updated. (See I-O status.)
20. If more than one file-name is specified in an OPEN statement, the result of executing this OPEN statement is the same as if a separate OPEN statement had been written for each file-name in the same order as specified in the OPEN statement. These separate OPEN statements would each have the same open mode specification, sharing-phrase, and timeout-phrase as specified in the OPEN statement.
21. The minimum and maximum record sizes for a file are established at the time the file is created and shall not subsequently be changed.
22. The SHARING phrase shall be specified only for shared files. (See Sharing mode.)
23. The SHARING phrase specifies what operations may be performed on the file associated with file-name-1 through other file connectors sharing the file. (See Sharing mode.)
24. The SHARING phrase overrides any SHARING clause in the file control entry of file-name-1. If there is no SHARING phrase on the OPEN statement, then file sharing is completely specified in the file control entry. If neither a SHARING phrase on the OPEN statement nor a SHARING clause in the file control entry is specified, then no sharing access controls are in place.
25. The RETRY phrase is used to control the behavior of an OPEN statement when the open mode or sharing mode requested conflicts with those of other file connectors that are currently associated with the file. The I-O status is set in accordance with the rules in RETRY phrase.
26. If the execution of the OPEN statement is unsuccessful, the physical file is not affected and the following actions take place in the following order:
    a. A value is placed in the I-O status associated with file-name to indicate the condition that caused the OPEN statement to be unsuccessful.
    b. Any applicable USE FOR EXCEPTION or USE AFTER EXCEPTION procedure is executed as specified for the rules for the USE statement.

PERFORM Statement

The PERFORM statement is used to transfer control explicitly to one or more procedures and to return control implicitly whenever execution of the specified procedure is complete. The PERFORM statement is also used to control execution of one or more imperative statements that are within the scope of that PERFORM statement.

General format

Format 1 (out-of-line):

PERFORM [[IN|OF] [EVENT] THREAD]
Procedure-name-1 [{THROUGH|THRU} procedure-name-2]
[HANDLE IN thread-handle-1]
[times-phrase | until-phrase | varying-phrase | timeout-phrase]

Format 2 (in-line):

PERFORM [[IN|OF] [EVENT] THREAD]
[HANDLE IN thread-handle-1]
[times-phrase | until-phrase | varying-phrase | timeout-phrase]
imperative-statement-1
[END-PERFORM]

Where timeout-phrase is:

TIMEOUT AFTER arithmetic-expression-1 SECONDS

where times-phrase is:

{identifier-1|integer-1} TIMES

where until-phrase is:

[WITH TEST {BEFORE|AFTER}] UNTIL condition-1

where varying-phrase is:

[WITH TEST {BEFORE|AFTER}]
VARYING {identifier-2|index-name-1} FROM {identifier-3|index-name-2|literal-1}
BY {identifier-4|literal-2} UNTIL condition-1
[AFTER {identifier-5|index-name-3} FROM {identifier-8|index-name-4|literal-3} BY
{identifier-7|literal-4} UNTIL condition-2] …

Syntax rules

1. If neither the TEST BEFORE nor the TEST AFTER phrase is specified, the TEST BEFORE phrase is assumed.
2. Each identifier represents a numeric elementary item described in the data division. Identifier-1 shall be an integer.
3. Each literal represents a numeric literal.
4. The words THROUGH and THRU are equivalent.
5. If an index-name is specified in the VARYING or AFTER phrase, then:
   a. The identifier in the associated FROM and BY phrases shall reference an integer data item.
   b. The literal in the associated FROM phrase shall be a positive integer.
   c. The literal in the associated BY phrase shall be a nonzero integer.
6. If an index-name is specified in the FROM phrase, then:
   a. The identifier in the associated VARYING or AFTER phrase shall reference an integer data item.
   b. The identifier in the associated BY phrase shall reference an integer data item.
   c. The literal in the associated BY phrase shall be an integer.
7. Literal in the BY phrase shall not be zero.
8. Condition-1, condition-2, ... , may be any conditional expression. (See Conditional expressions.)

9. Where procedure-name-1 and procedure-name-2 are both specified and either is the name of a procedure in the declaratives portion of the procedure division, both shall be procedure-names in the same declarative section.
10. At least six AFTER phrases shall be permitted in varying-phrase.
11. thread-handle-1 must be declared as USAGE HANDLE OF THREAD.

General rules

1. If an index-name is specified in the VARYING or AFTER phrase, and an identifier is specified in the associated FROM phrase, at the time the data item referenced by the identifier is used to initialize the index associated with the index-name, the data item shall have a positive value.
2. The END-PERFORM phrase delimits the scope of the in-line PERFORM statement. (See Scope of statements.)
3. An in-line PERFORM statement functions according to the following general rules for an otherwise identical out-of-line PERFORM statement, with the exception that the statements contained within the in-line PERFORM statement are executed in place of the statements contained within the range of procedure-name-1 (through procedure-name-2 if specified). Unless specially qualified by the word in-line or out-of-line, all the general rules that apply to the out-of-line PERFORM statement also apply to the in-line PERFORM statement.
4. When the PERFORM statement is executed, control is transferred to the first statement of the specified set of statements (except as indicated in general rules 9b, 9c, and 9d). This transfer of control occurs only once for each execution of a PERFORM statement. For those cases where a transfer of control to the specified set of statements does take place, an implicit transfer of control to the end of the PERFORM statement is established as follows:
    a. If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, the return is after the last statement of procedure-name-1.
    b. If procedure-name-1 is a section-name and procedure-name-2 is not specified, the return is after the last statement of the last paragraph in procedure-name-1.
    c. If procedure-name-2 is specified and it is a paragraph-name, the return is after the last statement of the paragraph.
    d. If procedure-name-2 is specified and it is a section-name, the return is after the last statement of the last paragraph in the section.
    e. If an in-line PERFORM statement is specified, an execution of the PERFORM statement is completed after the last statement contained within it has been executed.
5. There is no necessary relationship between procedure-name-1 and procedure-name-2 except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. In particular, GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point,

then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement, to which all of these paths shall lead.

6. If control passes to the specified set of statements by means other than a PERFORM statement, control will pass through the last statement of the set to the next executable statement as if no PERFORM statement referenced the set.

7. The PERFORM statements operate as follows:

   a. A PERFORM statement without times-phrase, until-phrase, or varying-phrase is the basic PERFORM statement. The specified set of statements referenced by this type of PERFORM statement is executed once and then control passes to the end of the PERFORM statement.

   b. If times-phrase is specified, the specified set of statements is performed the number of times specified by integer-1 or by the initial value of the data item referenced by identifier-1 for that execution. If at the time of the execution of a PERFORM statement, the value of the data item referenced by identifier-1 is equal to zero or is negative, control passes to the end of the PERFORM statement. Following the execution of the specified set of statements the specified number of times, control is transferred to the end of the PERFORM statement.

      During execution of the PERFORM statement, reference to identifier-1 shall not alter the number of times the specified set of statements is to be executed from that which was indicated by the initial value of the data item referenced by identifier-1.

   c. If until-phrase is specified, the specified set of statements is performed until the condition specified by the UNTIL phrase is true. When the condition is true, control is transferred to the end of the PERFORM statement. If the condition is true when the PERFORM statement is entered, and the TEST BEFORE phrase is specified or implied, no transfer to procedure-name-1 takes place, and control is passed to the end of the PERFORM statement. If the TEST AFTER phrase is specified, the PERFORM statement functions as if the TEST BEFORE phrase were specified except that the condition is tested after the specified set of statements has been executed. Any subscripting or reference modification associated with the operands specified in condition-1 is evaluated each time the condition is tested.

   d. If varying-phrase is specified, this variation of the PERFORM statement is used to augment the values referenced by one or more identifiers or index-names in an orderly fashion during the execution of a PERFORM statement. In the following discussion, every reference to identifier as the object of the VARYING, AFTER, and FROM (current value) phrases also refers to index-names. If identifier-2 or identifier-5 is subscripted, the subscripts are evaluated each time the content of the data item referenced by the identifier is set or augmented. If identifier-3, identifier-4, identifier-6, or identifier-7 is subscripted, the subscripts are evaluated each time the content of the data item referenced by the identifier is used in a setting or augmenting operation. Any subscripting or reference modification associated with the operands specified in condition-1 or condition-2 is evaluated each time the condition is tested.

8. The range of a PERFORM statement consists logically of all those statements that are executed as a result of executing the PERFORM statement through execution of the implicit transfer of control to the end of the PERFORM statement. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the PERFORM statement, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the PERFORM statement. The statements in the range of a PERFORM statement need not appear consecutively in the source program.
9. Statements executed as the result of a transfer of control caused by executing an EXIT PROGRAM statement are not considered to be part of the range of the PERFORM statement when:
    a. That EXIT PROGRAM statement is specified in the same program in which the PERFORM statement is specified, and
    b. The EXIT PROGRAM statement is within the range of the PERFORM statement.
10. Procedure-name-1 and procedure-name-2 shall not name sections or paragraphs in any other program in the run unit, irrespective of whether or not the other program contains or is contained within the program that includes the PERFORM statement. Statements in other programs in the run unit may only be obeyed as a result of executing a PERFORM statement, if the range of that PERFORM statement includes CALL and EXIT PROGRAM statements. (See Scope of names.)
11. If THREAD is specified, then the PERFORM area is done within a single, separate thread. All looping control is handled within the single, separate thread. When the control of the PERFORM is complete, the separate THREAD is automatically terminated. A PERFORM THREAD 10 TIMES means that the body of the PERFORM would be performed 10 times a single separate thread, not that 10 separate threads execute the body 1 or 10 times. Execution immediately continues with the statement after the PERFORM after the thread is started.
12. The results of executing the following sequence of PERFORM statements are undefined and no exception condition is set to exist when the sequence is executed:
    a. a PERFORM statement is executed and has not yet terminated, then
    b. within the range of that PERFORM statement another PERFORM statement is executed, then
    c. the execution of the second PERFORM statement passes through the exit of the first PERFORM statement.

    NOTE - Because this is undefined, the user should avoid such an execution sequence. On some implementations it causes stack overflows, on some it causes returns to unlikely places, and on others other actions can occur. Therefore, the results are unpredictable and are unlikely to be portable.

    Elastic COBOL supports a return-stack implementation of PERFORM.

2. PERFORM THREAD executes the perform statement in another thread, a reference to which may be stored in thread-handle-1.
3. PERFORM EVENT THREAD executes the perform statement on the event thread.
4. The timeout-phrase executes the perform until the timeout occurs, allowing the perform to be done for a certain period of time rather than a certain amount of iterations.

READ Statement

For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a specified record from a mass storage file.

General format

Format 1:

READ file-name-1 NEXT RECORD [INTO identifier-1]
[retry-phrase]
[IGNORING LOCK]
[WITH [NO | KEPT | IGNORE | WAIT] LOCK]
[WITH HOLD]
[FORMAT IS format-identifier-2 | format-literal-1]
[NULL-KEY-MAP IS null-key-identifier-5]
[NULL-MAP IS null-identifier-6]
[UNTIL condition-1]
[NO DATA imperative-statement-3]
[AT END imperative-statement-1]
[NOT AT END imperative-statement-2]
[END-READ]

Format 2:

READ file-name-1 [{NEXT | PREVIOUS | PRIOR | FIRST | LAST | {NEXT MODIFIED}} RECORD] [INTO identifier-1]
[retry-phrase]
[IGNORING LOCK]
[WITH [NO | KEPT | IGNORE | WAIT] LOCK]
[WITH HOLD]
[FORMAT IS format-identifier-2 | format-literal-1]
[NULL-KEY-MAP IS null-key-identifier-5]
[NULL-MAP IS null-identifier-6]
[UNTIL condition-1]
[INVALID KEY imperative-statement-1]
[NOT INVALID KEY imperative-statement-2]
[END-READ]

where retry-phrase is defined in RETRY phrase.

Syntax rules

ALL FORMATS

1. Condition-1 must be a valid condition, just as in the IF or PERFORM UNTIL statement.
2. PREVIOUS and PRIOR are synonymous.
3. PREVIOUS, PRIOR, FIRST, LAST, NEXT MODIFIED are not supported on all file systems.
4. NEXT MODIFIED is for AS/400 DDS files only. (DDS)

FORMAT 1

1. The INTO phrase may be specified in a READ statement:
   a. If no record description entry or only one record description is subordinate to the file description entry, or
   b. If the data item referenced by identifier-1 and all record-names associated with file-name-1 describe a group item or an elementary item of category alphanumeric or category national.
5. The storage area associated with identifier-1 and the record area associated with file-name-1 shall not be the same storage area.
6. If identifier-1 is a strongly typed group item, there shall be at most one record area subordinate to the FD for file-name-1. This record area, if specified, shall be a strongly typed group item of the same type as identifier-1.
7. The LOCK phrase shall not be specified in the same READ statement as the IGNORING LOCK phrase.
8. PREVIOUS is supported only for files of organization indexed.
9. WITH KEPT LOCK is the same as WITH LOCK.
10. WITH WAIT is the same as WITH LOCK and a retry forever.
11. WITH IGNORE LOCK is the same as IGNORING LOCK.
12. WITH HOLD is the same as WITH LOCK.
13. NO DATA is allowed only for TRANSACTION files. (DDS)

FORMAT 1

1. None of the phrases ADVANCING, AT END, NEXT, NOT AT END, or PREVIOUS shall be specified if ACCESS MODE RANDOM is specified in the file-control entry for file-name-1.
2. If neither the NEXT phrase nor the PREVIOUS phrase is specified and ACCESS MODE SEQUENTIAL is specified in the file-control entry for file-name-1, the NEXT phrase is implied.
3. If neither the NEXT phrase nor the PREVIOUS phrase is specified and ACCESS MODE DYNAMIC is specified in the file-control entry for file-name-1, the NEXT phrase is implied if any of the following phrases is specified:

   ADVANCING, AT END, or NOT AT END.

FORMAT 2

1. The KEY phrase shall be specified only if ORGANIZATION IS INDEXED is specified in the file-control entry for file-name-1.
2. Data-name-1 or record-key-name-1 shall be specified in the RECORD KEY clause or an ALTERNATE RECORD KEY clause associated with file-name-1.
3. Data-name-1 or record-key-name-1 may be qualified.

General rules

ALL FORMATS

1. The file connector referenced by file-name-1 shall be open in the input or I-O mode at the time this statement is executed.
2. The execution of the READ statement causes the value of the I-O status associated with file-name-1 to be updated. (See I-O status.)
3. When the logical records of a file are described with more than one record description, these records automatically share the same record area in storage; this is equivalent to an implicit redefinition of the area. The contents of any data items that lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.
4. The result of the successful execution of a READ statement with the INTO phrase is equivalent to the application of the following rules in the order specified:
   a. The execution of the same READ statement without the INTO phrase.
   b. The current record is moved from the record area to the area specified by identifier-1 according to the rules for the MOVE statement without the CORRESPONDING phrase. The size of the current record is determined by rules specified in the RECORD clause. If the file description entry contains a RECORD IS VARYING clause, the implied move is a group move. Item identification of the data item referenced by identifier-1 is done after the record has been read and immediately before it is moved to the data item. The record is available in both the record area and the data item referenced by identifier-1.

   If the execution of a READ statement with the INTO phrase is unsuccessful, the content of the data item referenced by identifier-1 is unchanged and item identification of the data item referenced by identifier-1 is not done.
5. The execution of a READ statement with the INTO phrase when there are no record description entries subordinate to the file description entry proceeds as though there were one record description entry describing a group item of the maximum size established by the RECORD clause.
6. If record locking is enabled for the file connector referenced by file-name-1 and the record identified for access by the general rules for the READ statement is locked by another file connector, the result of the operation depends on the presence or absence of the RETRY phrase. If the RETRY phrase is specified, additional attempts can made to read the record as specified in the rules in RETRY phrase. If the RETRY phrase is not specified or the record is not successfully accessed as specified by the RETRY phrase, the record operation

conflict condition exists. The I-O status is set in accordance with the rules for the RETRY phrase.

When the record operation conflict condition exists as a result of the READ statement:

a. The file position indicator is unchanged.

b. A value is placed into the I-O status associated with file-name-1 to indicate the record operation conflict condition.

c. The READ statement is unsuccessful.

7. If record locks are in effect and the execution of the READ statement is successful, the following actions take place:

   a. If single record locking is specified for the file connector associated with file-name-1, any record lock associated with that file connector is released at the completion of the successful execution of the READ statement.

   b. If multiple record locking is specified for the file connector associated with file-name-1, no record locks are released, except when the NO LOCK phrase is specified and the record accessed was already locked by that file connector. In this case, that record lock is released at the completion of the successful execution of the READ statement.

   c. If the lock mode is automatic, the record lock associated with the record accessed is set unless the NO LOCK phrase or the IGNORING LOCK phrase is specified on the READ statement.

   d. If lock mode is manual, the record lock associated with the record accessed is set only if the LOCK phrase is specified on the READ statement.

8. If the IGNORING LOCK phrase is specified on the READ statement, the requested record is made available, even if it is locked. When the IGNORING LOCK phrase is specified, no record lock is set by the execution of the READ statement, even when the lock mode is automatic.

9. If neither an at end nor an invalid key condition occurs during the execution of a READ statement, the AT END phrase or the INVALID KEY phrase is ignored, if specified, and the following actions occur:

   a. The I-O status associated with file-name-1 is updated and, if the record operation conflict condition did not occur, the file position indicator is set.

   b. If an exception condition that is not an at end or an invalid key condition exists, control is transferred according to the following rules:

      - If a USE AFTER EXCEPTION procedure is associated with the file connector referenced by file-name-1, control is transferred according to the rules for the specific exception condition and the rules for the USE statement following the execution of the associated declarative procedure.

      - If there is no USE AFTER EXCEPTION procedure associated with the file connector referenced by file-name-1, control is transferred according to the rules for the specific exception condition. If the exception condition is not a critical exception condition, control is transferred to the end of the READ statement.

c. If no exception condition exists, the record is made available in the record area and any implicit move resulting from the presence of an INTO phrase is executed. Control is transferred to the end of the READ statement or to imperative-statement-2, if specified. In the latter case, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement that 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.

10. If the number of character positions in the record that is read is less than the minimum size specified by the record description entries for file-name-1, the portion of the record area that is to the right of the last valid character read is undefined. If the number of character positions in the record that is read is greater than the maximum size specified by the record description entries for file-name-1, the record is truncated on the right to the maximum size. In either of these cases, the READ statement is successful and an I-O status is set indicating a record length conflict has occurred. (See I-O status.)
11. Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged; a record is available to the object program prior to the execution of imperative-statement-2 or imperative-statement-4, if specified, or prior to the execution of any statement following the READ statement, if imperative-statement-2 or imperative-statement-4 is not specified.
12. The END-READ phrase delimits the scope of the READ statement. (See Scope of statements.)
13. UNTIL condition-1 repeats the READ until condition-1 is true, or until the data file reaches an AT END or INVALID KEY condition. This is useful for positioning the file quickly to the given condition, allowing other fields related to the condition to be read.

FORMAT 1

1. An implicit or explicit NEXT phrase or a PREVIOUS phrase results in a sequential read: otherwise, the read is a random read and the rules for format 2 apply.
2. If the PREVIOUS phrase is specified, the file associated with the file connector referenced by file-name-1 shall be a single reel/unit mass storage file.
3. The setting of the file position indicator at the start of the execution of the READ statement is used in determining the record to be made available according to the following rules. Comparisons for records in sequential files relate to the record number. Comparisons for records in relative files relate to the relative key number. Comparisons for records in indexed files relate to the value of the current key of reference. For indexed files, the comparisons are made according to the collating sequence of the file.

    a. If the file position indicator indicates that no valid record position has been established, execution of the READ statement is unsuccessful.

b. If the file position indicator indicates that an optional input file is not present, execution proceeds as specified general rule 18.

c. If the file position indicator was established by a previous OPEN or START statement, the first existing record that is selected is either:

- If NEXT is specified or implied, the first existing record in the file whose record number or key value is greater than or equal to the file position indicator, or
- If PREVIOUS is specified, the first existing record in the file whose record number or key value is less than or equal to the file position indicator.

Note - For OPEN, this means that you normally get the first record in the file for sequential or relative and normally get an at end condition for indexed.

a. If the file position indicator was established by a previous READ statement and the file is a sequential or relative file, or an indexed file whose current key of reference does not allow duplicates, the first existing record in the file whose record number or key value is greater than the file position indicator if NEXT is specified or implied or is less than the file position indicator if PREVIOUS is specified is selected.

b. For indexed files, if the file position indicator was established by a previous READ statement, and the current key of reference does allow duplicates, the record that is selected is one of the following:

- If NEXT is specified or implied, the first record in the file whose key value is either equal to the file position indicator and whose logical position within the set of duplicates is immediately after the record that was made available by that previous READ statement, or whose key value is greater than the file position indicator.
- If PREVIOUS is specified, the first record in the file whose key value is either equal to the file position indicator and whose logical position within the set of duplicates is immediately prior to the record that was made available by that previous READ statement, or whose key value is less than the file position indicator.

If a record is found that satisfies this general rule and other general rules for the READ statement, the record is made available in the record area associated with file-name-1 unless the RELATIVE KEY clause is specified for file-name-1 and the number of significant digits in the relative record number of the selected record is larger than the size of the relative key data item. In that case, the execution of the READ statement is unsuccessful, the I-O status value is set to '14', the file position indicator is set to indicate that no next or previous logical record exists, and execution proceeds as specified in general rule 18.

If no record is found that satisfies the above rules, the file position indicator is set to indicate that no next or previous logical record exists and execution proceeds as specified in general rule 18. If a record is made available, the file position indicator is updated as follows:

a. For sequential files, the file position indicator is set to the record number of the record made available.

b. For relative files, the file position indicator is set to the relative record number of the record made available.

c. For indexed files, the file position indicator is set to the value of the current key of reference of the record made available.

4. If, during the execution of the READ statement, the end of reel/unit is recognized or a reel/unit contains no logical records, and the logical end of the file has not been reached for a given file connector, the following operations are executed:

   a. The standard ending reel/unit label procedure.

   b. A reel/unit swap. The current volume pointer is updated to point to the next reel/unit existing for the file.

   c. The standard beginning reel/unit label procedure.

18) If the file position indicator indicates that no next or previous logical record exists, or that an optional input file is not present, the following occurs in the order specified:

   a. A value, derived from the setting of the file position indicator, is placed into the I-O status associated with file-name-1 to indicate the at end condition. (See I-O status.)

   b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement-1. Any USE AFTER EXCEPTION procedure associated with the file connector referenced by file-name-1 is not executed. Execution then 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 in accordance with the rules of that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the READ statement and the NOT AT END phrase, if specified, is ignored.

   c. If the AT END phrase is not specified and a USE AFTER EXCEPTION procedure is associated with the file connector referenced by file-name-1, that procedure is executed. Control is then transferred to the end of the READ statement. The NOT AT END phrase is ignored if it is specified.

   d. If the AT END phrase is not specified and there is no USE AFTER EXCEPTION procedure associated with the file connector referenced by file-name-1, control is transferred to the end of the READ statement. The NOT AT END phrase is ignored if it is specified.

   When the at end condition occurs, execution of the READ statement is unsuccessful.

5. If a record operation conflict condition occurs during the execution of a sequential format READ statement, the content of the associated record area is undefined, and the key of reference for indexed files and the file position indicator remain unchanged. For any other unsuccessful execution of a READ statement, the content of the associated record area is undefined and the file position indicator is set to indicate that no valid record position has been established.

6. For a relative file, if the RELATIVE KEY clause is specified for file-name-1, the execution of a format 1 READ statement moves the relative record number of the record made available to the relative key data item according to the rules for the MOVE statement.
7. For an indexed file being sequentially accessed, records having the same duplicate value in an alternate record key that is the key of reference are made available in the same order, or, in the case of PREVIOUS, in the reverse order, in which they are released by execution of WRITE statements, or by execution of REWRITE statements that create such duplicate values.

FORMAT 2

1. If, at the time of the execution of a READ statement, the file position indicator indicates that an optional input file is not present, the invalid key condition exists and execution of the READ statement is unsuccessful, (See Invalid key condition.)
2. For a relative file, execution of a READ statement sets the file position indicator to the value contained in the data item referenced by the RELATIVE KEY clause for the file, and the record whose relative record number equals the file position indicator is made available in the record area associated with file-name-1. If the file does not contain such a record, the invalid key condition exists and execution of the READ statement is unsuccessful. (See Invalid key condition.)
3. For an indexed file accessed through a given file connector, if the KEY phrase is specified, data-name-1 or record-key-name-1 is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of sequential format READ statements for the file through the file connector until a different key of reference is established for the file through that file connector.
4. For an indexed file accessed through a given file connector, if the KEY phrase is not specified, the prime record key is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of sequential format READ statements for the file through the file connector until a different key of reference is established for the file through that file connector.
5. For an indexed file accessed through a given file connector, execution of a READ statement sets the file position indicator to the value in the key of reference. This value is compared with the value contained in the corresponding data item of the stored records in the file until the first record having an equal value is found. In the case of an alternate key with duplicate values, the first record found is the first record of a sequence of duplicates that was released to the MSCS. The record so found is made available in the record area associated with file-name-1. If no record is so identified, the invalid key condition exists and execution of the READ statement is unsuccessful. (See Invalid key condition.)

RECEIVE Statement

The RECEIVE statements receives messages sent by other threads or sessions.

General format

RECEIVE identifier-4 FROM {{THREAD thread-handle-1} | {LAST THREAD} | {ANY THREAD}}…

[{BEFORE TIME literal-1 | numeric-identifier-1}

{WITH TEST ONLY}

{WITH NO WAIT}

{THREAD IN thread-handle-2}

{SIZE OF identifier-2}

{STATUS IN identifier-3}]…

[ON EXCEPTION imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-RECEIVE]

Syntax rules

1. WITH TEST ONLY and WITH NO WAIT are equivalent to BEFORE TIME 0.
2. Thread-handle-1 and thread-handle-2 must be USAGE HANDLE OF THREAD data items.
3. Identifier-3 must be a two character group item.
4. Identifier-4 is the destination for the data received from the signal thread.

General rules

1. RECEIVE waits for another thread, the signal thread, to terminate or SEND a message.
2. The signal thread is the THREAD thread-handle-1 thread, the LAST thread, or for ANY THREAD, the first signal thread to terminate or send an appropriate message.
3. RECEIVE finishes immediately if there is already a message waiting for it.
4. A timeout in hundredths of a second may be specified by using BEFORE TIME, WITH TEST ONLY, or WITH NO WAIT. A timeout of 0 is effectively only tests the message status and returns. WITH TEST ONLY and WITH NO WAIT are therefore the same as BEFORE TIME 0.
5. The signal thread is stored in THREAD IN thread-handle-2.
6. The size of the message is stored in identifier-2.
7. The status codes are from the following:

00	Success
10	Thread Inactive Exception
99	Timeout Exception

8. If an exception occurred, imperative-statement-2 is executed. The flow of control then continues after the END-RECEIVE unless imperative-statement-2 transferred the flow of control.
9. If no exception occurred, imperative-statement-1 is executed. The flow of control then continues after the END-RECEIVE unless imperative-statement-1 transferred the flow of control.

10. SEND/RECEIVE/WAIT work across sessions as well as threads and may be used for inter-session communication on the same computer.
11. See SEND/RECEIVE.

RELEASE Statement

The RELEASE statement transfers records to the initial phase of a sort operation.

General format

RELEASE record-name-1 [FROM identifier-1]

Syntax rules

1. Record-name-1 shall be the name of a logical record in a sort-merge file description entry and it may be qualified.
2. If identifier-1 is a function-identifier, it shall reference an alphanumeric function. Record-name-1 and identifier-1 shall not reference the same storage area.
3. If record-name-1 is defined in a containing program, the file description entry for the file associated with record-name-1 shall contain a GLOBAL clause.

General rules

1. A RELEASE statement shall be executed only when it is within the range of an input procedure being executed by a SORT statement that references the file-name associated with record-name-1.
2. The execution of a RELEASE statement causes the record named by record-name-1 to be released to the initial phase of a sort operation.
3. The logical record released by the execution of the RELEASE statement is no longer available in the record area unless the sort-merge file-name associated with record-name-1 is specified in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as the file associated with record-name-1.
4. The result of the execution of a RELEASE statement with the FROM phrase is equivalent to the execution of the following statements in the order specified:
   a. The statement:

      MOVE identifier-1 TO record-name-1

      according to the rules specified for the MOVE statement.

   b. The same RELEASE statement without the FROM phrase.
5. After the execution of the RELEASE statement is complete, the information in the area referenced by identifier-1 is available, even though the information in the area referenced by record-name-1 is not available except as specified by the SAME RECORD AREA clause.
6. If the number of character positions to be released to the sort operation is greater than the number of character positions in record-name-1, the content of

the character positions that extend beyond the end of record-name-1 are undefined.

RETURN Statement

The RETURN statement obtains either sorted records from the final phase of a sort operation or merged records during a merge operation.

General format

RETURN file-name-1 RECORD [INTO identifier-1]
[AT END imperative-statement-1]
[NOT AT END imperative-statement-2]
[END-RETURN]

Syntax rules

1. The storage area associated with identifier-1 and the record area associated with file-name-1 shall not be the same storage area.
2. File-name-1 shall be described by a sort-merge file description entry in the data division.
3. The INTO phrase may be specified in a RETURN statement:
   a. If only one record description is subordinate to the sort-merge file description entry, or
   b. If all record-names associated with file-name-1 and the data item referenced by identifier-1 describe a group item or an elementary alphanumeric item.

General rules

1. A RETURN statement shall be executed only when it is within the range of an output procedure being executed by a MERGE or SORT statement that references file-name.
2. When the logical records in a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items that lie beyond the range of the current data record are undefined at the completion of the execution of the RETURN statement.
3. The execution of the RETURN statement causes the next existing record in the file referenced by file-name-1, as determined by the keys listed in the SORT or MERGE statement, to be made available in the record area associated with file-name-1. If no next logical record exists in the file referenced by file-name-1, the at end condition exists and control is transferred to imperative-statement-1 of the AT END phrase. Execution continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement 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 RETURN statement and the NOT AT END phrase is ignored, if specified. When the at end condition occurs, execution of the RETURN statement is unsuccessful and the contents of the record area associated with file-name-1

are undefined. After the execution of imperative-statement-1 in the AT END phrase, no RETURN statement may be executed as part of the current output procedure.

4. If an at end condition does not occur during the execution of a RETURN statement, then after the record is made available and after executing any implicit move resulting from the presence of an INTO phrase, control is transferred to imperative-statement-2, if specified; otherwise, control is transferred to the end of the RETURN statement.
5. The END-RETURN phrase delimits the scope of the RETURN statement. (See Scope of statements.)
6. The result of the execution of a RETURN statement with the INTO phrase is equivalent to the application of the following rules in the order specified:
   a. The execution of the same RETURN statement without the INTO phrase.
   b. The current record is moved from the record area to the area specified by identifier-1 according to the rules for the MOVE statement without the CORRESPONDING phrase. The size of the current record is determined by rules specified for the RECORD clause. If the file description entry contains a RECORD IS VARYING clause, the implied move is a group move. The implied MOVE statement does not occur if the execution of the RETURN statement was unsuccessful. Any subscripting associated with identifier-1 is evaluated after the record has been read and immediately before it is moved to the data item. The record is available in both the record area and the data item referenced by identifier-1.

REWRITE Statement

The REWRITE statement releases a logical record for an output or input-output file. It also used for vertical positioning of lines within a logical page.

The REWRITE statement is also used to control DDS files, including display files. (DDS)

General format

Format 1 (sequential):

REWRITE record-name-1 [FROM identifier-1 | literal-1]
[WITH HOLD]
[WITH NO LOCK]
[WITH KEPT LOCK]
[WITH WAIT]
[WITH IGNORE LOCK]
[ADVANCING ON LOCK]
[IGNORING LOCK]
[ALLOWING UPDATERS]
[WITH NO CONTROL]
[retry-phrase]
[NULL-MAP null-name]

[AT {END-OF-PAGE | EOP} imperative-statement-1]
[NOT AT {END-OF-PAGE | EOP} imperative-statement-2]
[END-WRITE]

Format 2 (random):

REWRITE record-name-1 [FROM identifier-1 | literal-1]
[WITH HOLD]
[WITH NO LOCK]
[WITH KEPT LOCK]
[WITH WAIT]
[WITH IGNORE LOCK]
[ADVANCING ON LOCK]
[IGNORING LOCK]
[ALLOWING UPDATERS]
[WITH NO CONTROL]
[retry-phrase]
[FORMAT format-name]
[NULL-KEY-MAP null-key-name]
[NULL-MAP null-name]
[INVALID KEY imperative-statement-1]
[NOT INVALID KEY imperative-statement-2]
[END-WRITE]

Format 3 (transaction subfile) (DDS)

REWRITE SUBFILE record-name-1 [FROM identifier-1 | literal-1]
[FORMAT format-name]
[TERMINAL terminal-name]
[{INDICATOR | INDICATORS | INDIC} indicator-name]
[INVALID KEY imperative-statement-1]
[NOT INVALID KEY imperative-statement-2]
[END-WRITE]

Where retry-phrase is defined in the RETRY phrase.

Syntax rules

1. If identifier-1 is a function-identifier, it shall reference an alphanumeric or national function. Record-name-1 and identifier-1 shall not reference the same storage area.
2. Record-name-1 is the name of a logical record in the file section of the data division and may be qualified.
3. The INVALID KEY and the NOT INVALID KEY phrases shall not be specified for a REWRITE statement that references a relative file in sequential access mode.
4. If record-name-1 is defined in a containing program, the file description entry for the file associated with record-name-1 shall contain a GLOBAL clause.
5. If record-name-1 is specified, identifier-1 or literal-1 shall be valid as a sending operand in a MOVE statement specifying record-name-1 as the receiving operand.

6. If file-name-1 is specified, literal-1 shall be an alphanumeric, Boolean, or national literal. A figurative constant shall not be specified.
7. File-name-1 shall not reference a report file or a sort-merge file description entry.
8. WITH KEPT LOCK is the same as WITH LOCK.
12. WITH WAIT is the same as WITH LOCK and a retry forever.
13. WITH IGNORE LOCK is the same as IGNORING LOCK.
14. WITH HOLD is the same as WITH LOCK.

Syntax rules (DDS)

1. Format-name is a nonnumeric identifier or literal; it must be uppercase and at most 10 characters.
2. Terminal-name is a nonnumeric identifier or literal.
3. Indicator-name is a numeric, numeric table, or group item.
4. Null-key-name is an identifier.
5. Null-name is an identifier.
6. Starting-line is an integer or integer identifier.

General rules

1. The file referenced by the file-name-1 or the file-name associated with record-name-1 shall be a mass storage file and shall be open in the I-O mode at the time of execution of this statement.
2. For file connectors in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of the REWRITE statement shall have been a successfully executed READ statement.

   The mass storage control system (MSCS) logically replaces the record that was accessed by the READ statement.
3. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area unless the file-name associated with record-name-1 is specified in a SAME RECORD AREA clause.

   The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as the file associated with record-name-1.
4. The result of the execution of a REWRITE statement specifying record-name-1 and the FROM phrase is equivalent to the execution of the following statements in the order specified:

   a. The statement:

      MOVE identifier-1 TO record-name-1

      or

      MOVE literal-1 TO record-name-1

      according to the rules specified for the MOVE statement.

   b. The same REWRITE statement without the FROM phrase.

5. The figurative constant SPACE when specified in the REWRITE statement references one alphanumeric space character.
6. The result of execution of a REWRITE statement with the FILE phrase is equivalent to the execution of the following in the order specified:
   a. The statement

      MOVE identifier-1 TO implicit-record-1

      or

      MOVE literal-1 TO implicit-record-1

   b. The statement

      REWRITE implicit-record-1

   where implicit-record-1 refers to the record area for file-name-1 and is treated:

   - when identifier-1 is a function-identifier, as though implicit-record-1 were a record description entry subordinate to the file description entry having the same category, the same length, and the same representation as the returned value of the function.
   - when identifier-1 is not a function-identifier, as though implicit-record-1 were a record description entry subordinate to the file description entry having the same description as identifier-1, or
   - when literal-1 is specified, as though implicit-record-1 were a record description entry subordinate to the file description entry having the same category, the same length, and the same representation as literal-1.
7. After the execution of the REWRITE statement is complete, the information in the area referenced by identifier-1 is available, even though the information in the area referenced by record-name-1 is not available except as specified by the SAME RECORD AREA clause.
8. If a record lock is associated with the record to be logically replaced in the file by this REWRITE statement, then:
   a. If the record lock is associated with the file connector referenced by file-name-1 or the file-name associated with record-name-1, the record lock is released at the successful completion of the execution of the REWRITE statement unless at least one of the following is true:
      - multiple record locking was specified and the NO LOCK phrase was not specified on the REWRITE statement, or
      - the LOCK phrase is specified on the REWRITE statement.
   b. If a different file connector has the record locked, the I-O status is set in accordance with the rules of RETRY phrase.
9. If single record locking was specified for the file connector referenced by file-name-1 or the file-name associated with record-name-1, any record locks for that file connector on records other than the one to be logically replaced by this REWRITE statement are released at the completion of the successful execution of the REWRITE statement.

10. If the LOCK phrase is specified and record locks have an effect for the file connector referenced by file-name-1 or the file-name associated with record-name-1, the record lock associated with the record accessed is set.
11. The file position indicator is not affected by the execution of a REWRITE statement.
12. The execution of the REWRITE statement causes the value of the I-O status of the file connector associated with file-name-1 or record-name-1 to be updated. (See I-O status.)
13. The execution of the REWRITE statement releases a logical record to the operating system.
14. When record-name-1 is specified, if the number of character positions to be written to the file is greater than the number of character positions in record-name-1, the content of the character positions that extend beyond the end of record-name-1 are undefined.
15. The END-REWRITE phrase delimits the scope of the REWRITE statement. (See Scope of statements.)

SEQUENTIAL FILES

If the number of character positions specified in literal-1, the data item referenced by identifier-1, or the record referenced by record-name-1 is not equal to the number of character positions in the record being replaced, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the content of the record area is unaffected and the I-O status of the file connector is set to a value indicating the cause of the condition. (See I-O status.)

RELATIVE AND INDEXED FILES

1. The number of character positions in the record referenced by identifier-1, literal-1, or the record referenced by record-name-1 may or may not be equal to the number of character positions in the record being replaced.
2. Transfer of control following the successful or unsuccessful execution of the REWRITE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the REWRITE statement. (See Invalid key condition.)
3. The number of character positions in literal-1, the data item referenced by identifier-1, or the record referenced by record-name-1 shall not be larger than the largest or smaller than the smallest number of character positions allowed by the RECORD IS VARYING clause associated with file-name-1 or the file-name associated with record-name-1. In either of these cases the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the contents of the record area are unaffected and the I-O status of file-name-1 or the file associated with record-name-1 is set to a value indicating the cause of the condition. (See I-O status.)

RELATIVE FILES

For a file accessed in either random or dynamic access mode, the mass storage control system (MSCS) logically replaces the record identified by the relative key data item specified for file-name-1 or the file-name associated with record-name-1. If the file does not contain the record specified by the key, the invalid key condition exists. When the invalid key condition is recognized, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the contents of the record area are unaffected and the I-O status of file-name-1 or the file-name associated with record-name-1 is set to a value indicating the cause of the condition. (See I-O status.)

INDEXED FILES

1. For a file in the sequential access mode, the record to be replaced is specified by the value of the prime record key. When the REWRITE statement is executed the value of the prime record key of the record to be replaced shall be equal to the value of the prime record key of the last record read from this file.
2. For a file in the random or dynamic access mode, the record to be replaced is specified by the prime record key.
3. Execution of the REWRITE statement for a record that has an alternate record key occurs as follows:
    a. When the value of a specific alternate record key is not changed, the order of retrieval when that key is the key of reference remains unchanged.
    b. When the value of a specific alternate record key is changed, the subsequent order of retrieval of that record may be changed when that specific alternate record key is the key of reference. When duplicate key values are permitted, the record is logically positioned last within the set of duplicate records containing the same alternate record key value as the alternate record key value that was placed in the record.
4. The invalid key condition exists under the following circumstances:
    a. When the file is open in the sequential access mode, and the value of the prime record key of the record to be replaced is not equal to the value of the prime record key of the last record read from the file connector, or
    b. When the file is open in the dynamic or random access mode, and the value of the prime record key of the record to be replaced is not equal to the value of the prime record key of any record existing in the file, or
    c. When the value of an alternate record key of the record to be replaced, for which duplicates are not allowed, equals the value of the corresponding data item of a record already existing in the file.
5. When the invalid key condition is recognized, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the content of the record area is unaffected and the I-O status of file-name-1 or the file-name associated with record-name-1 is set to a value indicating the cause of the condition. (See I-O status.)

General rules (DDS)

FORMAT phrase (DDS)

The FORMAT phrase is required when there are multiple records for the given file. Format-name is the record format name the I-O operation uses to select which record format to use. Format-name must be an uppercase character string of at most 10 characters. A format of SPACES is treated as if there were no FORMAT phrase.

INDICATOR phrase (File DDS)

NULL-KEY-MAP phrase (File DDS)

The NULL-KEY-MAP phrase specifies the null-byte map value for DATABASE files with the ALWNULL attribute.

NULL-MAP (File DDS)

The NULL-MAP phrase specifies the null-byte map value for the DATABASE file with the ALWNULL attribute.

STARTING AT LINE phrase (Display DDS)

The STARTING phrase specifies the starting line number for record formats using the variable starting line keyword.

The actual line number used is the starting line plus DDS starting line minus one.

ROLLING phrase (Display DDS)

The ROLLING phrase allows display lines to be moved, up or down. All lines vacated are cleared.

ROLLBACK Statement

The ROLLBACK statement undoes transactions done after the prior COMMIT or OPEN.

General format

ROLLBACK TRANSACTION

General rules

1. Rollback is currently ignored by the runtime.
2. This statement is treated as commentary.

SEARCH Statement

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

General format

Format 1 (serial):

SEARCH identifier-1 [VARYING {identifier-2|index-name-1}]
[AT END imperative-statement-1]
{WHEN condition-1 {imperative-statement-2|NEXT SENTENCE}}…
[END-SEARCH]

NOTE - NEXT SENTENCE is an archaic feature and its use should be avoided.

Format 2 (all):

SEARCH ALL identifier-1 [AT END imperative-statement-1]
WHEN {data-name-1 {IS EQUAL TO|IS =} {identifier-3|literal-1|arithmetic-expression-1} | condition-name-1}
[AND {data-name-2 {IS EQUAL TO|IS =} {identifier-4|literal-2|arithmetic-expression-2} | condition-name-2}]
{imperative-statement-2|NEXT SENTENCE}
[END-SEARCH]

NOTE - NEXT SENTENCE is an archaic feature and its use should be avoided.

Syntax rules

1. In both formats 1 and 2, identifier-1 shall not be subscripted or reference modified, but its description shall contain an OCCURS clause including an INDEXED BY phrase. The description of identifier-1 in format 2 shall also contain the KEY IS phrase in its OCCURS clause.
2. Identifier-2 shall reference a data item described as usage index or as a numeric elementary data item without any positions to the right of the assumed decimal point. Identifier-2 may not be subscripted by the first (or only) index-name specified in the INDEXED BY phrase in the OCCURS clause associated with identifier-1.
3. In format 1, condition-1 may be any conditional expression. (See Conditional expressions.)
4. In format 2, all referenced condition-names shall be defined as having only a single value. The data-name associated with a condition-name shall appear in the KEY IS phrase in the OCCURS clause associated with identifier-1. Each data-name-1, data-name-2 may be qualified. Each data-name-1, data-name-2, condition-name-1, condition-name-2 shall be subscripted by the first index-name associated with identifier-1 along with any subscripts required to uniquely reference data-name-1, data-name-2, condition-name-1, condition-name-2. Each data-name-1, data-name-2 shall be referenced in the KEY IS phrase in the OCCURS clause associated with identifier-1. Each condition-name-1, condition-name-2 shall be associated with a data-name that is referenced in the KEY IS phrase in the OCCURS clause associated with identifier-1. Identifier-3, identifier-4, or identifiers specified in arithmetic-expression-1, arithmetic-expression-2 shall not be referenced in the KEY IS phrase in the OCCURS clause associated with identifier-1 or be subscripted by the first index-name associated with identifier-1.

   In format 2, when a data-name in the KEY IS phrase in the OCCURS clause associated with identifier-1 is referenced, or when a condition-name associated with a data-name in the KEY IS phrase in the OCCURS clause associated with identifier-1 is referenced, all preceding data-names in the KEY IS phrase in the OCCURS clause referenced by identifier-1 or their associated condition-names shall also be referenced.

5. In format 2, the first index-name associated with identifier-1 shall not be followed by an operator + or - and an integer.
6. If the END-SEARCH phrase is specified, the NEXT SENTENCE phrase shall not be specified.

General rules

ALL FORMATS

1. The SEARCH statement automatically varies the first or only index associated with identifier-1 and tests conditions specified in WHEN phrases in the SEARCH statement to determine whether a table element satisfies these conditions. Any subscripting specified in a WHEN phrase is evaluated each time the conditions in that WHEN phrase are evaluated. For Format 1, an additional index or data item may be varied. If identifier-1 references a data item that is subordinate to a data item whose data description entry contains an OCCURS clause, only the setting of an index associated with identifier-1 (and any data item referenced by identifier-2 or any index referenced by index-name-1, if specified) is modified by the execution of the SEARCH statement. The subscript that is used to determine the occurrence of each superordinate table to search is specified by the user in the WHEN phrases. Therefore, each appropriate subscript shall be set to the desired value before the SEARCH statement is executed.

   Upon completion of the search operation, one of the following occurs:

   a. If the search operation is successful according to the general rules that follow, then: the search operation is terminated immediately; the index being varied by the search operation remains set at the occurrence number that caused a WHEN condition to be satisfied; if the WHEN phrase contains the NEXT SENTENCE phrase, control is transferred to an implicit CONTINUE statement immediately preceding the next separator period; if the WHEN phrase contains imperative-statement-2, control if transferred to that imperative-statement-2 and execution continues according to the rules for each statement specified in that imperative-statement-2. If the execution of a procedure branching or conditional statement results in an explicit transfer of control, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of that imperative-statement-2, control is transferred to the end of the SEARCH statement.

   b. If the search operation is unsuccessful according to the general rules that follow, then:

      - If the AT END phrase is specified, control is transferred to imperative-statement-1 and execution continues according to the rules for each statement specified in imperative-statement-1. If the execution of a procedure branching or conditional statement results in an explicit transfer of control, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the SEARCH statement.

      - If the AT END phrase is not specified and neither exception condition was raised because they were not enabled, control is transferred to the end of the SEARCH statement.

2. Any subscripting specified in a WHEN phrase is evaluated once at the start of each evaluation of the condition specified in that WHEN phrase.
3. The comparison associated with each WHEN phrase is executed in accordance with the rules specified for conditional expressions. (See Conditional expressions.)
4. The scope of a SEARCH statement may be terminated by any of the following:
    a. An END-SEARCH phrase at the same level of nesting.
    b. A separator period.
    c. An ELSE or END-IF phrase associated with a previous IF statement. (See Scope of statements.)

FORMAT 1

1. The index to be varied by the search operation is referred to as the search index and it is determined as follows:
    a. If the VARYING phrase is not specified, the search index is the index referenced by the first (or only) index-name specified in the INDEXED phrase in the OCCURS clause associated with identifier-1.
    b. If the VARYING identifier-2 phrase is specified, the search index is the same as in general rule 5a and the following also applies:
        - If identifier-2 references an index data item, that data item is incremented by the same amount as, and at the same time as, the search index.
        - If identifier-2 references an integer data item, that data item is incremented by the value one at the same time as the search index is incremented.
    c. If the VARYING index-name-1 phrase is specified, the search index depends on the following:
        - If index-name-1 is specified in the INDEXED BY phrase in the OCCURS clause associated with identifier-1, the index referenced by index-name-1 is the search index.
        - If index-name-1 is not one of the indexes specified in the INDEXED phrase in the OCCURS clause associated with identifier-1, the search index is the same as in general rule 5a. The index referenced by index-name-1 is incremented by one occurrence number at the same time as the search index is incremented.

        Only the data item and indexes indicated are varied by the search operation. All other indexes associated with identifier-1 are unchanged by the search operation.
2. The search operation is serial, starting from the occurrence number that corresponds to the value of the search index at the beginning of the execution of the SEARCH statement.
3. The number of occurrences of identifier-1, the last of which is permissible, is specified in the OCCURS clause.

If, at the start of the execution of the SEARCH statement, the search index contains a value that corresponds to an occurrence number that is not greater than the highest permissible occurrence number for identifier-1, the search operation proceeds by evaluating the conditions in the order they are written. If none of the conditions is satisfied, the search index is incremented by one occurrence number. The process is then repeated using the new index setting unless the new value for the search index corresponds to a table element outside the permissible range of occurrence values. If one of the conditions is satisfied upon its evaluation, the search operation is successful and the execution proceeds as indicated in general rule 1a.

FORMAT 2

1. At the start of the execution of a SEARCH statement with the ALL phrase specified, the following conditions shall be true. If any of the conditions are not true, the results of the search operation are undefined.

   a. The contents of the key data items that are referenced in the WHEN phrase shall be sufficient to identify a unique table element.

   b. The contents of each key data item referenced in the WHEN phrase shall be sequenced in the table according to the ASCENDING or DESCENDING phrase associated with that key data item. (See OCCURS clause.)

   c. If identifier-1 is subordinate to one or more Data Description entries that contain an OCCURS clause, the evaluation of the conditions within a WHEN phrase that reference a key data item subordinate to identifier-1 shall result in the same occurrence number for any subscripts associated with a given level of the superordinate tables. That is, the outermost level occurrence numbers shall all be equal, the next level occurrence numbers shall all be equal down to, but not including, the innermost table.

2. The search index is the index referenced by the first (or only) index-name specified in the INDEXED phrase in the OCCURS clause associated with identifier-1. Any other indexes associated with identifier-1 remain unchanged by the search operation.

3. A non-serial type of search operation may take place. The initial setting of the search index is ignored. Its setting is varied during the search operation in a manner specified by a binary search. At no time is it set to a value that exceeds the value which corresponds to the last element of the table or is less than the value which corresponds to the first element of the table. The length of the table is discussed in the OCCURS clause. If any of the conditions specified in the WHEN phrase cannot be satisfied for any setting of the search index within the permitted range, the final setting of the search index is undefined, the search operation is unsuccessful, execution proceeds as indicated in general rule 1b. If all the conditions can be satisfied, the search operation is successful and execution proceeds as indicated in general rule 1a.

SEEK Statement

Prepare to do a random read/write on a relative file. This statement is treated as commentary as it has no effect.

General format

SEEK file-name-1 RECORD

Syntax rules

file-name-1 must be open.

General rules

1. file-name-1 is prepared to do a read or write ahead of time by seeking to the correct position.
2. This statement is for compatibility only and is treated as commentary by Elastic COBOL.

SEND Statement

General format

SEND {literal-1 | identifier-1} TO {THREAD thread-handle-1 | LAST THREAD | ANY THREAD }…

Syntax rules

1. literal-1 or the contents of identifier-1 is the data to send to another thread.
2. Thread-handle-1 must be USAGE HANDLE OF THREAD.

General rules

1. The thread which does the SEND is known as the signal thread.
2. The signal thread sends a signal to the specified destination, which is either a particular thread, either thread-handle-1 or the last thread, or to any thread.
3. The signal will be received by the first thread to do a WAIT which listens to the signal thread. A WAIT on ANY THREAD will match a SEND TO ANY THREAD.
4. The data sent can be used for thread synchronization, or it can be actual data sent across the pipe.
5. SEND/RECEIVE/WAIT work across sessions as well as threads and may be used for inter-session communication on the same computer.
6. See RECEIVE/WAIT.

SESSION Statement

The SESSION statement starts a new child SESSION. A SESSION is similar to a THREAD, but there is a more complete separation of data. A new WORKING-STORAGE and FILE SECTION, including file references, is created. A SESSION may represent a single traditional COBOL program running transparently alongside other traditional COBOL programs within the same process. Some environments, such as the Servlet environment, automatically start a new SESSION for each client.

General format

Format 1

SESSION TO {procedure-name-1 | SELF | object-reference-1 }
[END-SESSION]

Syntax rules

1. Procedure-name-1 shall specify a procedure in the procedure division.
2. Object-reference-1 shall specify an object descending from java.lang.Thread or implementing java.lang.Runnable.

General rules

1. A separate thread of execution is started.
2. The separate thread of execution begins at procedure-name-1, at the beginning of the program for SELF, and at the run() method for object-reference-1.
3. A separate data and file area are allocated for the program when specifying procedure-name-1.

SET Statement

The SET statement provides a means for:

- establishing reference points for table handling operations by setting indexes associated with table elements,
- altering the status of external switches,
- altering the value of conditional variables,
- assigning object references,
- altering the attributes associated with a screen item,
- assigning the address of a data item to a data-pointer data item,
- assigning the address of a program to a program-pointer data item

General format

Format 1 (index-assignment):
SET {index-name-1|identifier-1} … TO {index-name-2|identifier-2|integer-1}

Format 2 (index-arithmetic):
SET {index-name-3} … {UP BY|DOWN BY} {identifier-3|integer-2}

Format 3 (switch-setting):
SET {{mnemonic-name-1}… TO {ON|OFF}}…

Format 4 (condition-setting):
SET {{condition-name-1} … TO {TRUE|FALSE}}…

Format 5 (object-identifier-assignment):
SET {identifier-5} … TO identifier-6

Format 6 (attribute):
SET screen-name-1 ATTRIBUTE

{ {BELL|BLINK|HIGHLIGHT|LOWLIGHT|REVERSE-VIDEO|UNDERLINE|AUTO|
BLANK LINE|BLANK SCREEN|ERASE EOL|ERASE EOS|FULL|SECURE|REQUIRED|
GRIDLINE|LEFTLINE|OVERLINE} {OFF|ON}}…

Format 7 (data-pointer-assignment):

SET {ADDRESS OF data-name-1|identifier-7} TO {ADDRESS OF data-name-2|
identifier-8|NULL|NULLS}

Format 8 (event-determination):

SET identifier-9 TO EVENT

Format 9 (exit status):

SET EXIT TO {TRUE|FALSE}

Format 10 (get object field value):

SET identifier-10 TO nonnumeric-literal-1 OF {identifier-11|class-name-1}

Format 11(set object field value):

SET nonnumeric-literal-1 OF {identifier-11|class-name-1} TO {identifier-10|literal-1}

Format 12(procedure pointer):

SET identifier-12 TO identifier-13

SET identifier-12 TO {ENTRY procedure-name|NULL|NULLS}

Format 13 (native object):

SET identifier-13 TO FUNCTION NATIVE(file-name-1|identifier-14)

Format 14 (applet hook):

SET SAME TO identifier-15

Format 15 (default constructor):

SET identifier-16 TO INITIALIZED

Format 16 (Get GUI property):

SET {identifier-17|identifier-19} TO basic-type nonnumeric-literal-2 OF identifier-18

Format 17 (Get GUI parsed property):

SET identifier-17 TO class-name-2 nonnumeric-literal-3 OF identifier-18

Format 18 (Set GUI property):

SET {[nonnumeric-literal-2 OF] identifier-18}
TO {basic-type literal-2}|{ class-name-2 literal-2}

Where basic-type is one of:

CHARACTER
BINARY-BYTE
BINARY-SHORT
BINARY
BINARY-LONG
BINARY-DOUBLE
FLOAT-SHORT
FLOAT-LONG
BIT

Format 19 (Boolean)

SET prim-boolean-id-1 TO { TRUE | FALSE}

Format 20 (thread priority):

SET THREAD [thread-handle-id] PRIORITY TO thread-priority

Format 21 (file-prefix)

SET FILE-PREFIX TO file-prefix

Format 22 (window columns)
SET SCREEN SIZE screen-size

Format 23 (window lines)
SET SCREEN LINES screen-lines

Format 24 (current window)
SET {{INPUT | INPUT-OUTPUT | I-O} | OUTPUT} WINDOW TO {CONSOLE | window-handle}

Format 25 (set environment)
SET ENVIRONMENT {env-name TO {set-env-value | NULL | NULLS } }

Format 26 (get environment)
SET get-env-value TO ENVIRONMENT env-name

Format 27 (set configuration)
SET CONFIGURATION {conf-name TO {set-conf-value | NULL | NULLS } }

Format 28 (get configuration)
SET get-conf-value TO CONFIGURATION conf-name

Format 29 (color attribute)
SET screen-name-1 ATTRIBUTE {FOREGROUND-COLOR | FOREGROUND-COLOUR | BACKGROUND-COLOR | BACKGROUND-COLOUR} TO color-number

Format 30 (object hash table)
SET object-hashtable (hash-key) TO {object-identifier | NULL | NULLS}

Format 31 (numeric hash table)
SET numeric-hashtable (hash-key) TO {[VALUE] numeric-identifier | NULL | NULLS}

Format 32 (alphanumeric hash table)
SET alphanumeric-hashtable (hash-key) TO {[VALUE] identifier | NULL | NULLS}

Format 33 (hash table assignment)
SET hashtable-1 TO {hashtable-2 | NULLS}

Syntax rules

1. Thread-priority is an integer or contents of a numeric identifier.
2. File-prefix is a literal or the contents of a nonnumeric data item.
3. Screen-size is an integer literal or the contents of an integer data item.
4. Screen-lines is an integer literal or the contents of an integer data item.
5. Env-name is a nonnumeric literal or contents of nonnumeric identifier.
6. Conf-name is a nonnumeric literal or contents of nonnumeric identifier.
7. Set-env-value is a literal or contents of identifier.
8. Get-env-value is an identifier.
9. Set-conf-value is a literal or contents of identifier.
10. Get-conf-value is an identifier.
11. NULL and NULLS are synonymous.

12. Hash-key is any literal or the contents of an identifier.

FORMATS 1 AND 2

1. All references to index-name-1, identifier-1, and index-name-3 apply equally to all repetitions thereof.
2. Identifier-1 and identifier-2 shall each reference an index data item or an elementary item described as an integer.
3. Identifier-3 shall reference an elementary numeric integer.
4. Integer-1 and integer-2 may be signed. Integer-1 shall be positive or zero.

FORMAT 3

Mnemonic-name-1 shall be associated with an external switch, the status of which may be altered. All external switches may be referenced by the SET statement.

FORMAT 4

1. Condition-name-1 shall be associated with a conditional variable.
2. If the FALSE phrase is specified, the FALSE phrase shall be specified in the VALUE clause of the data description entry for condition-name-1.

FORMAT 6

1. Identifier-6 shall be an object reference.
2. If identifier-5 is not object reference, then the value of its toString() method is copied to identifier-5 as if by a MOVE with a nonnumeric-literal.
3. If identifier-5 is an object reference, then identifier-6 must be assignable to identifier-5. This is always true if identifier-5 and identifier-6 are the same declared class-name, or both universal object references (java.lang.Object). This is also true if identifier-6's declared class-name inherits from identifier-5's declared class-name; this is always the case when identifier-5 declared as a universal object reference (java.lang.Object).

FORMAT 7

1. Any particular screen-attribute shall not be specified more than once in a SET statement.
2. HIGHLIGHT and LOWLIGHT shall not both be specified in the same SET statement.

FORMAT 8

1. Identifier-7 shall reference a data item of category data-pointer. Identifier-8 shall be of category data-pointer.
2. Data-name-1 shall be a based item.
3. If identifier-7 references a restricted data-pointer, identifier-8 shall reference a data-pointer restricted to the same type. If data-name-1 is a strongly typed item or a restricted pointer, identifier-8 shall reference a data-pointer restricted to the type of data-name-1.

If identifier-8 references a restricted data-pointer, either identifier-7 shall reference a data-pointer restricted to the same type or data-name-1 shall be a strongly typed item of the type to which identifier-8 is restricted.

FORMAT 9

Format 10 SET's shall only occur within a procedure started as a result of an EVENT.

FORMATS 10 and 11

1. Identifier-11 is an object-reference.
2. Class-name-1 is a static class reference.
3. Nonnumeric-literal-1 is the field name of the object-reference identifier-11 or static field name of class-name-1.

FORMAT 12

Identifier-12 and identifier-13 shall be of usage procedure pointer.

FORMAT 13

1. Identifier-13 shall be an object reference.
2. File-name-1 or identifier-14 shall be a COBOL primitive from which an object is obtained.

FORMAT 14

Identifier-15 shall be an object reference which shall be attached to the applet represented by this program.

FORMAT 15

Identifier-16 shall be an object reference defined as a universal object or as class-name with a default constructor (suitable for new classname()).

FORMATS 16, 17 and 18

1. Identifier-18 shall be an object reference.
2. Identifier-17 shall not be an object reference.
3. Identifier-19 shall be an object reference.
4. Nonnumeric-literal-2 shall be a property name; it should be specified without the 'get' or 'set' and the first character will automatically be uppercased, e.g., the getText() property has a nonnumeric-literal-2 of 'text'.
5. Class-name-2 specifies the format of nonnumeric-literal-3 for parsing/deparsing of text to object conversion.
6. Basic type conversion between Elastic COBOL and Java is CHARACTER to char, BINARY-BYTE to byte, BINARY-SHORT to short, BINARY and BINARY-LONG to int, BINARY-DOUBLE to long, FLOAT-SHORT to float, FLOAT-LONG to double, and BIT to Boolean.
7. Class-name-2 preceding a literal undergoes a parsing, where:
    a. For java.awt.Dimension, the literal is "(width,height)",
    b. For java.awt.Font, the literal is "fontname[[,[BOLD] [ITALIC] [PLAIN]],size=12]"

c. For java.awt.Color, the literal is “#RRGGBB” where RR is hex digits (00-FF) for red, GG is hex digits (00-FF) for green, and BB is hex digits (00-FF) for blue, or the literal is a color name “black”, “blue”, “cyan”, “darkGray”, “gray”, “green”, “lightGray”, “magenta”, “orange”, “pink”, “red”, “white”, or “yellow”.

d. For java.lang.String, the literal is the String itself,

e. For java.awt.BorderLayout, the literal is “(hgap,vgap)”,

f. For java.awt.CardLayout, the literal is “(hgap,vgap)”,

g. For java.awt.Point, the literal is “(x,y)”,

h. For java.awt.Cursor, the literal is “cursor-number”,

i. For java.io.File, the literal is the filename itself,

j. For java.awt.Insets, the literal is “(top,left,bottom,right)”,

k. For java.awt.Image, the literal is the image name,

l. For java.awt.Rectangle, the literal is “(x,y,w,h)”,

m. And for other classes a new default instance.

General rules

FORMATS 1 AND 2

1. Index-names are associated with a given table by being specified in the INDEXED BY phrase of the OCCURS clause for that table.

2. In format 1, the following action occurs:

   a. Index-name-1 is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element referenced by index-name-2, identifier-2, or integer-1. If identifier-2 references an index data item, the content of the data item referenced by identifier-2 is placed in the index referenced by index-name-1 unchanged. If index-name-2 is associated with the same table as index-name-1, the content of the index referenced by index-name-2 is placed in the index referenced by index-name-1 unchanged.

   b. If identifier-1 references an index data item, it may be set equal to either the content of index-name-2 or identifier-2 where identifier-2 also references an index data item.

   c. If identifier-1 does not reference an index data item, it may be set only to an occurrence number that corresponds to the value of index-name-2. Neither identifier-2 nor integer-1 may be used in this case.

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

3. In format 2, the content of index-name-3 is incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of integer-2 or the data item referenced by identifier-3; thereafter, the process is repeated for each recurrence of index-

name-3. For each repetition, the value of the data item referenced by identifier-3 is used as it was at the beginning of the execution of the statement.

4. Table 19, Validity of operand combinations on format 1 SET statements, shows the validity of each of the operand combinations in format 1 of the SET statement and the applicable general rules.

Validity of operand combinations on format 1 SET statements

Sending item	Receiving item		
	Integer data item	**Index**	**Index data item**
Integer literal	Invalid/2c	Valid/2a	Invalid/2b
Integer data item	Invalid/2c	Valid/2a	Invalid/2b
Index	Valid/2c	Valid/2a	Valid/2b
Index data item	Invalid/2c	Valid/2a	Valid/2b

FORMAT 3

The status of each external switch associated with the specified mnemonic-name-1 is modified such that the truth value resultant from evaluation of a condition-name associated with that switch will reflect an on status if the ON phrase is specified, or an off status if the OFF phrase is specified. (See Switch-status condition.)

FORMAT 4

1. The literal in the VALUE clause associated with condition-name-1 is placed in the conditional variable according to the rules of the VALUE clause (see VALUE clause). If more than one literal is specified in the VALUE clause, the conditional variable is set to the value of the first literal that appears in the VALUE clause.
2. If multiple condition-names are specified, the results are the same as if a separate SET statement had been written for each condition-name-1 in the same order as specified in the SET statement.
3. If the FALSE phrase is specified, the literal in the FALSE phrase of the VALUE clause associated with condition-name-1 is placed in the conditional variable according to the rules for the VALUE clause.

FORMAT 6

If identifier-6 is a predefined object identifier or an object reference, a reference to the object identified by identifier-6 is placed into identifier-5.

FORMAT 7

When the SET statement is executed, the settings of the specified attributes of the screen item referenced by screen-name-1 are turned on or off as specified. The latest settings of the attributes are used when screen-name-1 is referenced in an ACCEPT screen or DISPLAY screen statement.

FORMAT8

1. If identifier-7 is specified, the address identified by identifier-8 is stored in the data item referenced by identifier-7.
2. If data-name-1 is specified, the address identified by identifier-8 is assigned to the based item referenced by data-name-1.

FORMAT 9

If identifier-9 is an object reference, then the event's object is set according to the rules for setting one object to another object. If identifier-9 is not an object

reference, then the name of the event as specified in the EVENT FOR clause is copied to identifier-9 as if by a MOVE.

FORMAT 10

1. SET EXIT TO TRUE sets the default behavior when all threads are done to exit.
2. SET EXIT TO FALSE sets the default behavior when all threads are done to not exit.
3. The default behavior is initially true. If displaying only to SYSOUT/SYSIN and running a single-threaded traditional COBOL program, exit will occur automatically at the end of the program.
4. The default behavior is set to false by creating THREAD, by using the GUI CONSOLE for DISPLAY/ACCEPT, and by using AWT 1.0 GUI verbs.
5. When false, the GUI CONSOLE will display Finished when the main thread is done if visible, but the program will not end until the user clicks the close gadget.
6. The GUI CONSOLE is created upon first use.

Format 19

Set the value of a primitive Boolean to either true or false.

Format 20

Thread-priority is the execution priority of the current or given thread. The higher the number, the higher the priority. Thread priority is defined by the Java Virtual Machine.

Format 21

Set the FILE-PREFIX configuration variable to file-prefix. This syntax is for compatibility only; the preferred method of setting the FILE-PREFIX is with SET CONFIGURATION "FILE-PREFIX".

Format 22

Set the screen size in columns of the default window.

Format 23

Set the screen size in lines of the default window.

Format 24

1. Set window-handle to be the current and possibly the active window. The current window is used for DISPLAY statements. The active window is used for DISPLAY and user ACCEPT. Window-handle must be a valid window, already created by a DISPLAY statement.
2. INPUT, INPUT-OUTPUT, and I-O cause window-handle to be the current and active window.
3. OUTPUT causes window-handle to become the current window.

Format 25

Set the System Property represented by env-name to set-env-value, or remove it using NULL or NULLS.

Format 26 (set environment)

Get the System Property represented by env-name into get-env-value.

Format 27 (get configuration):

Set the configuration parameter conf-name to set-conf-value, or remove it using NULL or NULLS.

Format 28 (set configuration):

Get the configuration parameter env-name into get-conf-value.

Format 29 (color attribute):

Set the screen-name-1 foreground or background color attribute to the given color-number dynamically.

Formats 30-32

1. Set a hashtables keyed value to a reference to another variable or a value. This allows any access as hashtable('key') to refer to the original variable or to the value. The type of the value is defined by the hashtable's type, either object, numeric, or alphanumeric. An object hashtable can contain any object. A numeric hashtable can contain numeric values and variables. An alphanumeric hashtable can contain any alphanumeric or numeric class value or variable; it will always be treated as alphanumeric class.
2. Hashtable access is very efficient and is a good substitute for pointers; it is both safer and allows for named indexing.
3. A hashtable's keyed value may be removed by setting it to NULL or NULLS.
4. A variable being placed into an alphanumeric or numeric hashtable may be forced to be treated as a value, evaluated before placement into the table, by using the keyword VALUE. Unless using VALUE, the original variable reference is used, allowing the hash-key to alias the original variable.

Format 33 (hashtable assignment)

1. Set hashtable-1 to be the same as hashtable-2 or clear it.
2. Setting hashtable-1 to NULL or NULLS clears the hashtable, removing all hash-keys and their entries.
3. Setting hashtable-1 to hashtable-2 makes hashtable-1 alias hashtable-2; this means that all keys in hashtable-2 are now accessible through hashtable-1, and all entries accessible from hashtable-1 are no longer accessible through hashtable-1.
4. If assigning a hashtable of one type to another type, all references must be valid; generally, any may be used from an object-hashtable, and numeric entries may be used when referenced from an alphanumeric hashtable.
5. A hashtable("key") may be used any place a literal is accepted of the given type.

Example use of hashtable types:

Hasher program:

```
identification division.
program-id. hasher.

data division.
working-storage section.

01 num numeric-hashtable.
01 alp alphanumeric-hashtable.
```

```
01 my-1 pic 999 value 1.
01 my-2 pic 999 value 2.
01 my-3 pic 999 value 3.
01 my-4 pic 999 value 4.
01 my-5 pic 999 value 5.

01 my-a1 pic xxx value "abc".
01 my-a2 pic xxx value "bcd".
01 my-a3 pic xxx value "cde".
01 my-a4 pic xxx value "def".
01 my-a5 pic xxx value "efg".

procedure division.
main-paragraph.
    set num("one") to my-1
    set num("two") to my-2
    set num("three") to my-3
    set num("four") to my-4
    set num("five") to my-5

    set alp("one") to my-a1
    set alp("two") to my-a2
    set alp("three") to my-a3
    set alp("four") to my-a4
    set alp("five") to my-a5

    display "num:" upon sysout
    display num("one") "," num("two") "," num("three") "," num("four") "," num("five") upon sysout

    display "alp:" upon sysout
    display alp("one") "," alp("two") "," alp("three") "," alp("four") "," alp("five") upon sysout
    .

    move alp("one") to alp("two")
    move num("one") to num("two")
    display "now alp two=" alp("two") upon sysout
    .
```

Hasher output:

```
num:
001,002,003,004,005
alp:
abc,bcd,cde,def,efg
now alp two=abc
```

SHOW Statement

Show a graphical component. (AWT) (JFC) (GfxScreen)

General format

SHOW awt-id-1 | jfc-id-1 | {SCREEN {gfxscreen-1 | handle-id-1}} [UPON PRINTER]
END-SHOW

Syntax rules

1. awt-id-1 is an AWT graphics component. The component must have already been built.

2. jfc-id-1 is an object reference extending java.awt.Component. The component must have already been instantiated. This is the same as invoking "setVisible" using FALSE.
3. gfxscreen-1 is a graphical screen section component identifier. The component must have been previously displayed.
4. handle-id-1 is a HANDLE reference to a graphical screen section component. The component must have been previously displayed.

General rules

The graphical component is rendered invisible.

1. UPON PRINTER renders the component to the printer.
2. Do not SHOW a Frame referenced by awt-id; always BUILD it after hiding it. This is because a Frame has system resources which are allocated and deallocated by BUILD and HIDE.
3. BUILD WITH NO SHOW the awt-id if intending only to show it upon the printer.
4. SHOW is the inverse of HIDE.

SORT Statement

The SORT statement causes a set of records or table elements to be arranged in a user-specified sequence.

General format

```
SORT file-name-1 {ON {ASCENDING|DESCENDING} KEY {data-name-1}…}…
[WITH DUPLICATES IN ORDER]
{INPUT PROCEDURE IS
        procedure-name-1 [{THROUGH|THRU} procedure-name-2] |
USING {file-name-2}…}
{OUTPUT PROCEDURE IS
        procedure-name-3 [{THROUGH|THRU} procedure-name-4] |
GIVING {file-name-3}…}
```

Syntax rules

1. A SORT statement may appear anywhere in the procedure division except that a format 1 SORT statement shall not appear in the declarative portion.
2. Data-name-1 may be qualified.
3. Data-name-1 shall not reference a group item that contains a variable-occurrence data item.
4. File-name-1 shall be described in a sort-merge file description entry in the data division.
5. If the USING phrase is specified and the file description entry for file-name-1 describes variable-length records, the file description entry for file-name-2 shall not describe records smaller than the smallest record nor larger than the largest record described for file-name-1. If the file description entry for file-name-1

describes fixed-length records, the file description entry for file-name-2 shall not describe a record that is larger than the record described for file-name-1.

6. Data-name-1 is a key data-name. Key data-names are subject to the following rules:
   a. The data items identified by key data-names shall be described in records associated with file-name-1.
   b. Key data items may be qualified.
   c. Key data items shall not be of the class Boolean or object.
   d. The data items identified by key data-names shall not be group items that contain variable-occurrence data items.
   e. If file-name-1 has more than one record description, then the data items identified by key data-names need be described in only one of the record descriptions. The same character positions that are referenced by a key data-name in one record description entry are taken as the key in all records of the file.
   f. None of the data items identified by key data-names may be described by an entry that either contains an OCCURS clause or is subordinate to an entry that contains an OCCURS clause.
   g. If the file referenced by file-name-1 contains variable-length records, all the data items identified by key data-names shall be contained within the first x character positions of the record, where x equals the minimum record size specified for the file referenced by file-name-1.
7. The words THRU and THROUGH are equivalent.
8. File-name-2 and file-name-3 shall be described in a file description entry that is not for a report file and is not a sort-merge file description entry.
9. If file-name-3 references an indexed file, the first specification of data-name-1 shall be associated with an ASCENDING phrase and the data item referenced by that data-name-1 shall occupy the same character positions in its record as the data item associated with the prime record key for that file.
10. No pair of file-names in the same SORT statement may be specified in the same SAME SORT AREA or SAME SORT-MERGE AREA clause. File-names associated with the GIVING phrase may not be specified in the same SAME AREA clause.
11. If the GIVING phrase is specified and the file description entry for file-name-3 describes variable-length records, the file description entry for file-name-1 shall not describe records smaller than the smallest record nor larger than the largest record described for file-name-3. If the file description entry for file-name-3 describes fixed-length records, the file description entry for file-name-1 shall not describe a record that is larger than the record described for file-name-1.

General rules

ALL FORMATS

1. The words ASCENDING and DESCENDING are transitive across all occurrences of data-name-1 until another word ASCENDING or DESCENDING is encountered.
2. The data items referenced by the specifications of data-name-1 are the key data items that determine the order in which records are returned from the file referenced by file-name-1 or the order in which the table elements are stored after sorting takes place. The order of significance of the keys is the order in which they are specified in the SORT statement, without regard to their association with ASCENDING or DESCENDING phrases.
3. If the DUPLICATES phrase is specified and the contents of all the key data items associated with one data record or table element are equal to the contents of the corresponding key data items associated with one or more other data records or table elements, the order of return of these records is:
    a. The order of the associated input files as specified in the SORT statement. Within a given input file the order is that in which the records are accessed from that file.
    b. The order in which these records are released by an input procedure, when an input procedure is specified.
    c. The relative order of the contents of these table elements before sorting takes place.
4. If the DUPLICATES phrase is not specified and the contents of all the key data items associated with one data record or table element are equal to the contents of the corresponding key data items associated with one or more other data records or table elements, the order of return of these records or the relative order of the contents of these table elements is undefined.
5. The alphanumeric collating sequence that applies to the comparison of key data items of class alphabetic and class alphanumeric, and the national collating sequence that applies to the comparison of key data items of class national, are each separately determined at the beginning of the execution of the SORT statement by the collating sequence established as the program collating sequence.
6. If the file referenced by file-name-1 contains only fixed-length records, any record in the file referenced by file-name-2 containing fewer character positions than that fixed-length is space filled on the right with alphanumeric space characters beginning with the first character position after the last character in the record when that record is released to the file referenced by file-name-1.
7. To determine the relative order in which two records are returned from the file referenced by file-name-1, the contents of corresponding key data items are compared according to the rules for comparison of operands in a relation condition, starting with the most significant key data item.
    a. If the contents of the corresponding key data items are not equal and the key is associated with the ASCENDING phrase, the record containing the key data item with the lower value is returned first;
    b. If the contents of the corresponding key data items are not equal and the key is associated with the DESCENDING phrase, the record containing the key data item with the higher value is returned first; and,

c. If the contents of the corresponding key data items are equal, the determination is made on the contents of the next most significant key data item.

8. The execution of the SORT statement consists of three distinct phases as follows:

   a. Records are made available to the file referenced by file-name-1. If INPUT PROCEDURE is specified, the execution of RELEASE statements in the input procedure makes the records available. If USING is specified, implicit READ and RELEASE statements make the records available.. When this phase terminates, the file referenced by file-name-2 is not in an open mode.

   b. The file referenced by file-name-1 is sequenced. No processing of the files referenced by file-name-2 and file-name-3 takes place during this phase.

   c. The records of the file referenced by file-name-1 are made available in sorted order. The sorted records are either written to the file referenced by file-name-3 or, by the execution of a RETURN statement, are made available for processing by the output procedure. When this phase terminates, the file referenced by file-name-3 is not in the open mode.

9. 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, INVOKE, 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.

10. If an input procedure is specified, control is passed to the input procedure before the file referenced by file-name-1 is sequenced by the SORT statement. The compiler inserts a transfer of control mechanism at the end of the last statement in the input procedure and when control passes the last statement in the input procedure, the records that have been released to the file referenced by file-name-1 are sorted.

11. If the USING phrase is specified, all the records in the file(s) referenced by file-name-2 are transferred to the file referenced by file-name-1. For each of the files referenced by file-name-2 the execution of the SORT statement causes the following actions to be taken:

    a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the INPUT phrase and without a SHARING phrase had been executed. The absence of the SHARING phrase means that the sharing mode is completely determined by the SHARING clause, if any, in the file control entry for the file connector referenced by file-name-2.

    b. The logical records are obtained and released to the sort operation. Each record is obtained as if a READ statement with the NEXT and the AT END phrases had been executed. If the file referenced by file-name-1 is described with variable-length records, the size of any record released to file-name-1 is the size of that record when it was read from file-name-2, regardless of the content of the data item referenced by the DEPENDING

ON phrase of either a RECORD IS VARYING clause or an OCCURS clause specified in the sort-merge file description entry for file-name-1.

c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed. This termination is performed before the file referenced by file-name-1 is sequenced by the SORT statement. For a relative file, the content of the relative key data item associated with file-name-2 is undefined after the execution of the SORT statement if file-name-2 is not referenced in the GIVING phrase.

These implicit functions are performed such that any associated USE AFTER EXCEPTION/ERROR procedures are executed; however, the execution of such a USE procedure shall not cause the execution of any statement manipulating the file referenced by, or accessing the record area associated with, file-name-2.

The value of the data item referenced by the DEPENDING ON phrase of a RECORD IS VARYING clause specified in the file description entry for file-name-2 is undefined upon completion of the SORT statement.

12. The output procedure may consist of any procedure needed to select, modify, or copy the records that are made available one at a time by the RETURN statement in sorted order from the file referenced by file-name-1. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the output procedure, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the output procedure.

13. If an output procedure is specified, control passes to it after the file referenced by file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last statement in the output procedure and when control passes the last statement in the output procedure, the return mechanism provides for termination of the sort and then passes control to the next executable statement after the SORT statement. Before entering the output procedure, the sort procedure reaches a point at which it selects the next record in sorted order when requested. The RETURN statements in the output procedure are the requests for the next record.

14. If the GIVING phrase is specified, all the sorted records are written on the file referenced by file-name-3 as the implied output procedure for the SORT statement. For each of the files referenced by file-name-3, the execution of the SORT statement causes the following actions to be taken:

    a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase had been executed. This initiation is performed after the execution of any input procedure.

    b. The sorted logical records are returned and written onto the file. The records are written as if a WRITE statement without any optional phrases had been executed. If the file referenced by file-name-3 is described with variable-length records, the size of any record written to file-name-3 is the size of that record when it was read from file-name-1, regardless of the content of the data item referenced by the DEPENDING ON phrase of either a RECORD IS VARYING clause or an OCCURS clause specified in the file description entry for file-name-3.

For a relative file, the relative key data item for the first record returned contains the value '1'; for the second record returned, the value '2', etc. After execution of the SORT statement, the content of the relative key data item indicates the last record returned to the file.

c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

These implicit functions are performed such that any associated USE AFTER EXCEPTION/ERROR procedures are executed; however, the execution of such a USE procedure shall 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 EXCEPTION procedure associated with the file connector referenced by file-name-3 is executed; if control is returned from that USE procedure or if no such USE procedure is specified, the processing of the file is terminated as in general rule 14c above.

The value of the data item referenced by the DEPENDING ON phrase of a RECORD IS VARYING clause specified in the sort-merge file description entry for file-name-1 is undefined upon completion of the SORT statement for which the GIVING phrase is specified.

15. If the file referenced by file-name-3 contains only fixed-length records, any record in the file referenced by file-name-1 containing fewer character positions than that fixed-length is space filled on the right with alphanumeric space characters beginning with the first character position after the last character in the record when that record is returned to the file referenced by file-name-3.

START Statement

The START statement provides a basis for logical positioning within a file, for subsequent sequential retrieval of records.

General format

Format 1

START file-name-1
[KEY relational-operator {data-name-1|record-key-name-1}]
[INVALID KEY imperative-statement-1]
[NOT INVALID KEY imperative-statement-2]
[END-START]

where relational-operator is:

IS EQUAL TO
IS =
IS GREATER THAN
IS >
IS NOT LESS THAN OR EQUAL TO
IS NOT <=
IS NOT LESS THAN
IS NOT <

IS GREATER THAN OR EQUAL TO

IS >=

IS LESS THAN

IS <

IS LESS THAN OR EQUAL TO

IS <=

IS NOT GREATER THAN

IS NOT >

IS NOT GREATER THAN OR EQUAL TO

IS NOT >=

Format 2

START TRANSACTION

Syntax rules

1. File-name-1 shall be the name of a file connector that has either sequential or dynamic access.
2. File-name-1 shall be of organization relative or indexed.
3. The last eight relational-operators (<,<=) are available only for indexed files for use with READ PREVIOUS.
4. Data-name-1 or record-key-name-1 may be qualified.
5. For relative files, data-name-1, if specified, shall be the data item specified in the RELATIVE KEY phrase in the ACCESS MODE clause of the associated file control entry.
6. For indexed files, data-name-1, if specified, shall reference either:
   a. A data item specified as a record key associated with file-name-1 (see ALTERNATE RECORD KEY clause, and RECORD KEY clause), or
   b. Any data item of category alphanumeric or national whose leftmost character position within a record of the file corresponds to the leftmost character position of a record key associated with file-name-1 and defined without using the record-key-name syntax. The length of data-name-1 shall be less than or equal to the length of that record key in alphanumeric or national character positions, respectively.
7. Record-key-names-1 shall reference a record key associated with file-name-1. (See ALTERNATE RECORD KEY clause, and RECORD KEY clause.)

General rules

Format 1

1. The file connector referenced by file-name-1 shall be open in the input or I-O mode at the time that the START statement is executed.
2. The execution of the START statement does not alter either the content of the record area or the content of the data item referenced by the data-name specified in the DEPENDING ON phrase of the RECORD clause associated with file-name-1.

3. The execution of the START statement does not detect, acquire, or release record locks.
4. The execution of the START statement causes the value of the I-O status associated with file-name-1 to be updated. (See I-O status.)
5. If, at the time of the execution of the START statement, the file position indicator indicates that an optional input file is not present, the invalid key condition exists and the execution of the START statement is unsuccessful.
6. Transfer of control following the successful or unsuccessful execution of the START operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the START statement as specified in Invalid key condition.)
7. Following the unsuccessful execution of a START statement, the file position indicator is set to indicate that no valid next record has been established. For indexed files, the key of reference is undefined.
8. The END-START phrase delimits the scope of the START statement. (See Scope of statements.)

Format 2

1. START TRANSACTION is currently ignored by the Elastic COBOL runtimes and is present only for compatibility.

RELATIVE FILES

1. If the KEY phrase is omitted, the START statement behaves as though KEY IS EQUAL TO data-name-1 had been specified, where data-name-1 is the name of the key specified in the RELATIVE KEY clause associated with file-name-1.
2. The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file-name-1 and a data item as specified in general rule 10. Numeric comparison rules apply. (See Comparison of numeric operands.)
    a. If the relational operator is EQUAL, GREATER, NOT LESS, or GREATER OR EQUAL, the file position indicator is set to the relative record number of the first logical record in the file whose key satisfies the comparison searching the file sequentially.
    b. If the relational operator is LESS, NOT GREATER, or LESS OR EQUAL, the file position indicator is set to the relative record number of the first logical record in the file whose key satisfies the comparison searching the file in reverse order.
    c. If the comparison is not satisfied by any record in the file, the invalid key condition exists and the execution of the START statement is unsuccessful.
3. START with < or <= is not permitted on relative files.

INDEXED FILES

1. Arithmetic-expression-1 shall evaluate to a positive non-zero integer that is less than or equal to the length of the associated key.

2. If the KEY phrase is not specified, the behavior is the same as if KEY IS EQUAL TO data-name-1 or record-key-name-1 had been specified, with data-name-1 or record-key-name-1 being the prime record key for the file.
3. The key specified in the KEY phrase, or that shares a leftmost character with the data item specified in the KEY phrase, becomes the key of reference. This key of reference is used to establish the ordering of records for the purpose of this START statement; and, if the execution of the START statement is successful, the key of reference is also used for subsequent sequential READ statements.
4. Execution of the START statement behaves as if:
    a. The specified key is set up by moving the relevant parts of the record area into a temporary data area.
    b. The length of this temporary area is considered to be the length of record-key-name-1, if specified, or else the length of data-name-1.
    c. If the relational operator is EQUAL, GREATER, NOT LESS, or GREATER OR EQUAL, the file is searched sequentially with the key of reference being extracted from each record in turn into another temporary area. This second temporary area is truncated to the same length as the first.
    d. If the relational operator is LESS, NOT GREATER, or LESS OR EQUAL, the file is searched in reverse order with the key of reference being extracted from each record in turn into another temporary area. This second temporary area is truncated to the same length as the first.
    e. The comparison specified by the relational operator in the KEY phrase is made between these two temporary areas, with the second temporary area on the left hand side, and according to the collating sequence of the file. Comparison proceeds as specified for items of the class of data-name-1 and operands of equal length in Comparison of alphanumeric operands, or Comparison of national operands. Then, either:
        - The file position indicator is set to the value of the key of reference in the first logical record whose key satisfies the comparison, or
        - If the comparison is not satisfied by any record in the file, the invalid key condition exists and the execution of the START statement is unsuccessful.

STOP Statement

The STOP statement causes termination of the execution of the run unit.

General format

Format 1:

STOP {RUN
[WITH {ERROR|NORMAL} STATUS |
{GIVING|RETURNING} integer-1]} literal-2

Format 2:

STOP ALL RUN

Format 3:

STOP THREAD [object-reference-1]

Format 4 (stop handle)

STOP {THREAD handle-id} | {LAST THREAD}

Syntax rules

1. If a STOP RUN statement appears in a consecutive sequence of imperative statements within a sentence, it shall appear as the last statement in that sequence.
2. Identifier-1 shall reference an integer data item or a data item with usage display or usage national.
3. If literal-1 is numeric, it shall be an integer.
4. Object-reference-1 shall inherit from java.lang.Thread.
5. handle-id must be a USAGE HANDLE OF THREAD data item.

General rules

1. Execution of the run unit terminates and control is transferred to the operating system.
2. During the execution of a STOP RUN statement, an implicit CLOSE statement without any optional phrases is executed for each file that is in the open mode in the run unit. Any USE procedures associated with any of these files are not executed.
3. If the ERROR phrase is specified, the operating system will indicate an abnormal or error termination of the run unit if such a capability exists within the operating system.
4. If the NORMAL phrase is specified, the operating system will indicate a normal termination of the run unit if such a capability exists within the operating system.
5. If neither the ERROR phrase nor the NORMAL phrase is specified, the termination of the run unit is normal or specified by the RETURN-CODE register contents.
6. During execution of the STOP statement with literal-1 or identifier-1 specified, literal-1 or the contents of the data item referenced by identifier-1 are passed to the operating system.
7. Specifying literal-2 is obsolete and its use should be avoided. STOP literal-2 prompts the user with literal-2 to continue to execution or stop execution. The program parameter CONTINUE-RUN-MESSAGE may be used to override the Continue Run prompt.
8. STOP ALL RUN stops all threads.
9. STOP THREAD without object-reference-1 stops the current thread; in a single-threaded program, this stops the program.
10. STOP THREAD with an object-reference performs the stop() method on the thread.

11. In a GUI or multi-threaded environment, STOP RUN does a STOP THREAD on the current thread; in a non-GUI traditional environment, STOP RUN does a traditional STOP RUN.
12. STOP THREAD handle-id stops the specific thread referenced by handle-id; this thread will generally have been obtained through a PERFORM THREAD HANDLE IN or CALL THREAD HANDLE IN.
13. STOP LAST THREAD will stop the last created thread. Avoid this usage, present only for compatibility. Always specify the thread to stop directly.

STRING Statement

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

General format

STRING {{identifier-1|literal-1}… [DELIMITED BY {identifier-2|literal-2|SIZE}]}…
INTO identifier-3
[WITH POINTER identifier-4]
[ON OVERFLOW imperative-statement-1]
[NOT ON OVERFLOW imperative-statement-2]
[END-STRING]

Syntax rules

1. Identifier-1 or literal-1 shall not be indexes, pointers, or objects.
2. Literal-1 or literal-2 shall not be a figurative constant that begins with the word ALL.
3. All literals shall be described as alphanumeric or national literals, and all identifiers, except identifier-4, shall be described implicitly or explicitly as usage display or national. If any one of literal-1, literal-2, identifier-1, identifier-2, or identifier-3 is of class national, then all shall be of class national.
4. Identifier-3 shall not be reference modified.
5. Identifier-3 shall not represent an edited data item and shall not be described with the JUSTIFIED clause.
6. Identifier-4 shall be described as an elementary numeric integer data item of sufficient size to contain a value equal to 1 plus the size of the data item referenced by identifier-3. The symbol 'P' may not be used in the PICTURE character-string of identifier-4.
7. Where identifier-1 or identifier-2 is an elementary numeric data item, it shall be described as an integer without the symbol 'P' in its PICTURE character-string.
8. The DELIMITED phrase may be omitted only immediately preceding the INTO phrase. If it is omitted, DELIMITED BY SIZE is implied.

General rules

1. Identifier-1 or literal-1 represents the sending item. Identifier-3 represents the receiving item.

2. Literal-2 or the content of the data item referenced by identifier-2 indicates the character(s) delimiting the move. If the SIZE phrase is used, the content of the complete data item defined by identifier-1 or literal-1 is moved. When a figurative constant is used as the delimiter, it is a single-character literal of the same class as identifier-3.
3. When a figurative constant is specified as literal-1 or literal-2, it refers to an implicit one character data item whose usage shall be the same as the usage of identifier-3, either display or national.
4. When the STRING statement is executed, the transfer of data is governed by the following rules:
    a. Characters from literal-1 or from the content of the data item referenced by identifier-1 are transferred to the data item referenced by identifier-3 in accordance with the MOVE statement rules for alphanumeric-to-alphanumeric moves or, when identifier-3 is of class national, national-to-national moves, except that no space filling is provided.
    b. If the DELIMITED phrase is specified without the SIZE phrase, the content of the data item referenced by identifier-1, or the value of literal-1, is transferred to the receiving data item in the sequence specified in the STRING statement beginning with the leftmost character positions and continuing from left to right until the end of the sending data item is reached or the end of the receiving data item is reached or until the character(s) specified by literal-2, or by the content of the data item referenced by identifier-2, are encountered. The character(s) specified by literal-2 or by the data item referenced by identifier-2 are not transferred.
    c. If the DELIMITED phrase is specified with the SIZE phrase, the entire content of literal-1, or the content of the data item referenced by identifier-1, is transferred, in the sequence specified in the STRING statement, to the data item referenced by identifier-3 until all data has been transferred or the end of the data item referenced by identifier-3 has been reached. This behavior is repeated until all occurrences of literal-1 or data items referenced by identifier-1 have been processed.
5. If the POINTER phrase is specified, the data item referenced by identifier-4 shall be set to an initial value greater than zero prior to the execution of the STRING statement.
6. If the POINTER phrase is not specified, the following general rules apply as if the user had specified identifier-4 referencing a data item with an initial value of 1.
7. When characters are transferred to the data item referenced by identifier-3, the moves behave as though the characters were moved one at a time from the source into the character positions of the data item referenced by identifier-3 designated by the value of the data item referenced by identifier-4 (provided the value of the data item referenced by identifier-4 does not exceed the length of the data item referenced by identifier-3), and then the data item referenced by identifier-4 was increased by one prior to the move of the next character or prior to the end of execution of the STRING statement. The value of the data item referenced by identifier-4 is changed during execution of the STRING statement only by the behavior specified above.

8. At the end of execution of the STRING statement, only the portion of the data item referenced by identifier-3 that was referenced during the execution of the STRING statement is changed. All other portions of the data item referenced by identifier-3 will contain data that was present before this execution of the STRING statement.
9. Before each move of a character to the data item referenced by identifier-3, if the value associated with the data item referenced by identifier-4 is either less than one or exceeds the number of character positions in the data item referenced by identifier-3, the following occurs:
    a. No further data is transferred to the data item referenced by identifier-3.
    b. If the ON OVERFLOW phrase is not specified, the applicable Declarative, if any, is executed. If control is returned from that Declarative or there is no applicable Declarative, control is transferred to the end of the STRING statement and any NOT ON OVERFLOW phrase, if specified, is ignored.
    c. If the ON OVERFLOW phrase is specified, control is transferred to imperative-statement-1 and 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 in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the STRING statement.
10. If, at the time of execution of a STRING statement with the NOT ON OVERFLOW phrase, the conditions described in general rule 9 are not encountered, after completion of the transfer of data according to the other general rules, the ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the STRING statement or, if the NOT ON OVERFLOW phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the STRING statement.
11. The END-STRING phrase delimits the scope of the STRING statement. (See Scope of statements.)
12. 1If identifier-1, or identifier-2, occupies the same storage area as identifier-3, or identifier-4, or if identifier-3 and identifier-4 occupy the same storage area, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See Overlapping operands.)

SUBTRACT Statement

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

General format

Format 1 (simple):

SUBTRACT {identifier-1|literal-1} … FROM {identifier-2 [ROUNDED]} …
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-SUBTRACT]

Format 2 (giving):

SUBTRACT {identifier-1|literal-1} … FROM {identifier-2|literal-2}
GIVING {identifier-3 [ROUNDED]}…
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-SUBTRACT]

Format 3 (corresponding):

SUBTRACT {CORRESPONDING|CORR} identifier-1 FROM identifier-2 [ROUNDED]
…
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-SUBTRACT]

Format 4 (table)

SUBTRACT {TABLE | MULTIPLE} source-table FROM destination-table [ROUNDED]
[FROM INDEX source-start TO source-end]
[DESTINATION INDEX destination-start]
[ON SIZE ERROR statement-1]
[NOT ON SIZE ERROR statement-2]
[END-ADD]

Syntax rules

1. Each identifier shall refer to a numeric elementary item except that:
   a. In format 2, each identifier following the word GIVING shall refer to either an elementary numeric item or an elementary numeric-edited item.
   b. In format 3, each identifier shall refer to a group item.
2. Each literal shall be a numeric literal.
3. The composite of operands is determined by using all of the operands in the statement.
   a. In format 1 the composite of operands is determined by using all of the operands in a given statement.
   b. In format 2 the composite of operands is determined by using all of the operands in a given statement excluding the data items that follow the word GIVING.
   c. In format 3 the composite of operands is determined separately for each pair of corresponding data items.
4. The words CORR and CORRESPONDING are equivalent.

5. The identifiers may be primitive numerics, but ON SIZE ERROR is not supported in that case.
6. Additional rules and explanations relative to this statement are given in Scope of statements; Incompatible data; ROUNDED phrase; ON SIZE ERROR phrase and size error condition; Arithmetic statements; Overlapping operands; Multiple results in arithmetic statements; and CORRESPONDING phrase.

Format 4

1. Source-table and destination-table must both be tables consisting of numeric items.
2. Source-start and source-end must reference valid indices within source-table.
3. Destination-start must reference a valid index within destination-table.

General rules

1. When format 1 is used, the initial evaluation shall consist of determining the value to be subtracted, which is literal-1 or the value of the data item referenced by identifier-1, or if more than one is specified, the sum of such operands. The initial evaluation is subtracted from the value of the data item referenced by identifier-2 and the result shall be stored as the new value of the data item referenced by identifier-2.
2. When format 2 is used, the initial evaluation shall consist of determining the value to be subtracted, which is literal-1 or the value of the data item referenced by identifier-1, or if more than one is specified, the sum of such operands; and subtracting this value from literal-2 or the value of the data item referenced by identifier-2.
3. When format 3 is used, data items in identifier-1 shall be subtracted from and stored in corresponding items in identifier-2.
4. The compiler insures enough places are carried so as not to lose significant digits during execution.

Format 4

1. Across a range of indices, a table element in source-table is subtracted from a table element in destination-table.
2. Source-start specifies the start of the source range, defaulting to 1.
3. Source-end specifies the final element of the source range, defaulting to the final declared element of source-table.
4. Destination-start specifies the start of the destination range, defaulting to 1.

THREAD Statement

The THREAD statement causes a new thread of execution to begin control in another part of the procedure division; the original thread of execution containing the THREAD statement continues onwards.

General format

Format 1 (unconditional):

THREAD TO {procedure-name-1|SELF|object-reference-1}
[END-THREAD]

Syntax rules

1. Procedure-name-1 shall specify a procedure in the procedure division.
2. Object-reference-1 shall specify an object descending from java.lang.Thread or implementing java.lang.Runnable.

General rules

1. A separate thread of execution is created.
2. This separate thread of execution begins at procedure-name-1, at the beginning of the current program for SELF, and at the run() method for object-reference-1.

UN-EXCLUSIVE Statement

The UN-EXCLUSIVE statement releases exclusive access to a file by unlocking all records in the file. This statement must occur only on a file in the open state.

General format

UN-EXCLUSIVE file-name-1

Syntax rules

1. File-name-1 must be a valid file reference to a sharable file.
2. File-name-1 must be in the open state.
3. File-name-1 must have previously been set exclusive using the EXCLUSIVE verb.
4. The record server must be running. The exclusive/un-exclusive pair share files in the open mode rather than unopened mode, so the recordserver is used to lock all records.

UNLOCK Statement

The UNLOCK statement explicitly releases any record locks associated with a file connector, or releases critical section locks.

General format

Format 1

UNLOCK file-name-1 [RECORD|RECORDS]

Format 2

UNLOCK ALL [RECORD|RECORDS]

Format 3

UNLOCK [ALL] THREAD

Format 4

UNLOCK WAIT identifier

Syntax rules

File-name-1 shall not refer to a sort file or merge file.

General rules

Format 1

1. Any record locks associated with the file connector referenced by file-name-1 are released at the completion of the successful execution of the UNLOCK statement. The presence or absence of any record locks does not affect the success of the execution of the UNLOCK statement.

   NOTE: To unlock one particular record, use the READ statement with the NO LOCK phrase.

5. File-name-1 shall reference a file connector in the open mode.
6. The execution of the UNLOCK statement causes the value of the I-O status of the file connector referenced by file-name-1 to be updated.

Format 2

All records locked are released by an UNLOCK in format 2.

Format 3

Exit the critical section obtained by LOCK THREAD using UNLOCK THREAD, or LOCK ALL THREAD by UNLOCK ALL THREAD. See comments in LOCK THREAD.

Format 4

Notify a thread with LOCK WAIT identifier that it may continue by using UNLOCK WAIT identifier. See comments in LOCK THREAD.

UNSTRING Statement

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

General format

UNSTRING identifier-1
[DELIMITED BY [ALL] {identifier-2|literal-1} [OR [ALL] {identifier-3|literal-2}]…]
INTO {identifier-4 [DELIMITER IN identifier-5] [COUNT IN identifier-6]} …
[WITH POINTER identifier-7]
[TALLYING IN identifier-8]
[ON OVERFLOW imperative-statement-1]
[NOT ON OVERFLOW imperative-statement-2]
[END-UNSTRING]

Syntax rules

1. Literal-1 and literal-2 shall be literals of the category alphanumeric or national and shall not be a figurative constant that begins with the word ALL.
2. Identifier-1, identifier-2, identifier-3, and identifier-5 shall reference data items described implicitly or explicitly as category alphanumeric or category national.

3. If any of identifier-1, identifier-2, identifier-3, identifier-4, identifier-5, literal-1, or literal-2 are of category national, then all shall be of category national.
4. Identifier-4 shall be described implicitly or explicitly as usage display and category alphabetic, alphanumeric, or numeric; or as usage national and category national or numeric. Numeric items shall not be specified with picture character string 'P'.
5. Identifier-6 and identifier-8 shall reference integer data items (except that the symbol 'P' may not be used in the PICTURE character-string).
6. Identifier-7 shall be described as an elementary numeric integer data item of sufficient size to contain a value equal to 1 plus the size of the data item referenced by identifier-1. The symbol 'P' may not be used in the PICTURE character-string of identifier-7.
7. The DELIMITER IN phrase and the COUNT IN phrase may be specified only if the DELIMITED BY phrase is specified.
8. Identifier-1 shall not be reference modified.

General rules

1. All references to identifier-2 and literal-1 apply equally to identifier-3 and literal-2, respectively, and all recursions thereof.
2. The data item referenced by identifier-1 represents the sending area.
3. The data item referenced by identifier-4 represents the receiving area for data. The data item referenced by identifier-5 represents the receiving area for delimiters.
4. Literal-1 or the data item referenced by identifier-2 specifies a delimiter.
5. The data item referenced by identifier-6 represents the count of the number of characters within the data item referenced by identifier-1 isolated by the delimiters for the move to the data item referenced by identifier-4. This value does not include a count of the delimiter character(s).
6. The data item referenced by identifier-7 contains a value that indicates a relative character position within the area referenced by identifier-1.
7. The data item referenced by identifier-8 is a counter that is incremented by 1 for each occurrence of the data item referenced by identifier-4 accessed during the UNSTRING operation.
8. When a figurative constant is used as the delimiter, it stands for a single-character national literal if identifier-1 is a national data item; otherwise, it stands for a single-character alphanumeric literal.

   When the ALL phrase is specified, one occurrence or two or more contiguous occurrences of literal-1 (figurative constant or not) or the content of the data item referenced by identifier-2 are treated as if they were only one occurrence, and one occurrence of literal-1 or the data item referenced by identifier-2 is moved to the receiving data item according to the rules in general rule 12d.
9. When any examination encounters two contiguous delimiters, the current receiving area shall be space-filled if it is described as alphabetic, alphanumeric, or national; or zero-filled if it is described as numeric.

10. Each literal-1 or the data item referenced by identifier-2 represents one delimiter. When a delimiter contains two or more characters, all of the characters shall be present in contiguous positions of the sending item, and in the order given, to be recognized as a delimiter.

11. When two or more delimiters are specified in the DELIMITED BY phrase, an OR condition exists between them. Each delimiter is compared to the sending field. If a match occurs, the character(s) in the sending field is considered to be a single delimiter. No character(s) in the sending field shall be considered a part of more than one delimiter.

    Each delimiter is applied to the sending field in the sequence specified in the UNSTRING statement.

12. When the UNSTRING statement is initiated, the current receiving area is the data item referenced by identifier-4. Data is transferred from the data item referenced by identifier-1 to the data item referenced by identifier-4 according to the following rules:

    a. If the POINTER phrase is specified, the string of characters referenced by identifier-1 is examined beginning with the relative character position indicated by the content of the data item referenced by identifier-7. If the POINTER phrase is not specified, the string of characters is examined beginning with the leftmost character position.

    b. If the DELIMITED BY phrase is specified, the examination proceeds left to right until a delimiter specified by either literal-1 or the value of the data item referenced by identifier-2 is encountered. (See general rule 10.) If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current receiving area. However, if the sign of the receiving item is defined as occupying a separate character position, the number of characters examined is one less than the size of the current receiving area. Size is defined as number of character positions.

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

    c. The characters thus examined, excluding any delimiting characters, shall be treated as an elementary national data item if identifier-1 is of category national, and otherwise as an elementary alphanumeric data item, and shall be moved into the current receiving area according to the rules for the MOVE statement.

    d. If the DELIMITER IN phrase is specified the delimiting character(s) shall be treated as an elementary national data item if identifier-1 is of category national, and otherwise as an elementary alphanumeric data item and shall be moved into the data item referenced by identifier-5 according to the rules for the MOVE statement.

       If the delimiting condition is the end of the data item referenced by identifier-1, then the data item referenced by identifier-5 is space filled.

    e. If the COUNT IN phrase is specified, a value equal to the number of characters thus examined, excluding any delimiter characters, shall be

moved into the area referenced by identifier-6 according to the rules for an elementary move.

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

g. After data is transferred to the data item referenced by identifier-4, the current receiving area is the data item referenced by the next recurrence of identifier-4. The behavior described in general rules 12b through 12f is repeated until either all the characters are exhausted in the data item referenced by identifier-1, or until there are no more receiving areas.

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

14. The content of the data item referenced by identifier-7 will be incremented by one for each character examined in the data item referenced by identifier-1. When the execution of an UNSTRING statement with a POINTER phrase is completed, the content of the data item referenced by identifier-7 will contain a value equal to the initial value plus the number of characters examined in the data item referenced by identifier-1.

15. When the execution of an UNSTRING statement with a TALLYING phrase is completed, the content of the data item referenced by identifier-8 contains a value equal to its value at the beginning of the execution of the statement plus a value equal to the number of identifier-4 receiving data items accessed during execution of the statement.

16. Either of the following situations causes an overflow condition:

    a. An UNSTRING is initiated, and the value in the data item referenced by identifier-7 is less than 1 or greater than the number of character positions described for the data item referenced by identifier-1.

    b. If, during execution of an UNSTRING statement, all receiving areas have been acted upon, and the data item referenced by identifier-1 contains characters that have not been examined.

17. When an overflow condition exists, the following occurs:

    a. The UNSTRING operation is terminated.

    b. If the ON OVERFLOW phrase is not specified, the applicable declarative, if any, is executed. If control is returned from that declarative or there is no applicable declarative, control is transferred to the end of the UNSTRING statement and any NOT ON OVERFLOW phrase, if specified, is ignored.

    c. If the ON OVERFLOW phrase is specified, control is transferred to imperative-statement-1 and 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 in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the UNSTRING statement.

18. The END-UNSTRING phrase delimits the scope of the UNSTRING statement. (See Scope of statements.)

19. If, at the time of execution of an UNSTRING statement, the conditions described in general rule 16 are not encountered, after completion of the transfer of data according to the other general rules, the ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the UNSTRING statement or, if the NOT ON OVERFLOW phrase is specified, to imperative-statement-2. If control is transferred to imperative-statement-2, execution continues according to the rules for each statement specified in imperative-statement-2. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of imperative-statement-2, control is transferred to the end of the UNSTRING statement.

20. If identifier-1, identifier-2, or identifier-3, occupies the same storage area as identifier-4, identifier-5, identifier-6, identifier-7, or identifier-8, or if identifier-4, identifier-5, or identifier-6, occupies the same storage area as identifier-7 or identifier-8, or if identifier-7 and identifier-8 occupy the same storage area, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See Overlapping operands.)

USE Statement

The USE statement specifies

1. Procedures for error or exception handling that are in addition to the standard procedures provided by other facilities such as the input-output control system and the operating system.
2. Procedures that are executed after the detection of exception conditions.

General format

USE [GLOBAL] AFTER STANDARD {EXCEPTION|ERROR} PROCEDURE ON {{file-name-1}…|INPUT|OUTPUT|I-O|EXTEND}

Syntax rules

ALL FORMATS

1. A USE statement, when present, shall immediately follow a section header in the declaratives portion of the procedure division and shall appear in a sentence by itself. The remainder of the section shall consist of zero, one, or more procedural paragraphs that define the procedures to be used.
2. File-name-1, class-name-1, or interface-name-1 shall not be the same as an exception-name.
3. File-name-1 shall not be a sort or a merge file.
4. Procedure-names associated with a USE statement may be referenced in a different declarative section or in a non-declarative procedure only with a PERFORM statement.

FORMAT 1

1. The same file-name shall not appear in more than one USE AFTER EXCEPTION statement within the same procedure division.
2. The words ERROR and EXCEPTION are synonymous and may be used interchangeably.
3. The files implicitly or explicitly referenced in the USE statement need not all have the same organization or access.
4. The INPUT, OUTPUT, I-O, and EXTEND phrases may each be specified only once in the declaratives portion of a given procedure division.

General rules

ALL FORMATS

1. The USE statement is never executed; it merely defines the conditions calling for the execution of the USE procedures.
7. A declarative is invoked when any of the conditions described in the USE statement that prefaces the declarative occurs while the program is being executed. Only a declarative within the separately-compiled program that contains the statement that caused the qualifying condition is invoked when any of the conditions described in the USE statement that prefaces the declarative occurs while that separately-compiled program is being executed. If no qualifying declarative exists in the separately-compiled program, no declarative is executed.
8. For a given exception condition, only one qualifying declarative procedure is selected for execution.
9. For source elements contained within other source elements, multiple declaratives may be eligible for selection for a given exception condition. The declarative selected for execution is determined in the following order of precedence:
    a. the qualifying declarative in the source element that contains the statement that caused the condition to exist,
    b. a qualifying declarative with the GLOBAL attribute in the next inclusive directly containing source element. This step is repeated with the next higher directly containing source element until a declarative is selected or the outermost source element is reached.
10. If no qualifying declarative is found, none is executed.
11. Within a USE procedure, there shall not be the execution of any statement that would cause the execution of a USE procedure that had previously been invoked and had not yet returned control to the invoking routine.
12. Within a given procedure division, a USE statement specifying file-name-1 takes precedence over any USE statements specifying an INPUT, OUTPUT, I-O, or EXTEND phrase.
13. The procedures associated with a USE statement are executed by the input-output control system after completion of the standard input-output exception routine upon the unsuccessful execution of an input-output operation unless an AT END or INVALID KEY phrase takes precedence. The rules concerning when the procedures are executed are as follows:

a. If file-name-1 is specified, the associated procedure is executed when the condition described in the USE statement occurs.

b. If INPUT is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the input mode or in the process of being opened in the input mode, except those files referenced by file-name-1 in another USE statement specifying the same condition.

c. If OUTPUT is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the output mode or in the process of being opened in the output mode, except those files referenced by file-name-1 in another USE statement specifying the same condition.

d. If I-O is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the I-O mode or in the process of being opened in the I-O mode, except those files referenced by file-name-1 in another USE statement specifying the same condition.

e. If EXTEND is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the extend mode or in the process of being opened in the extend mode, except those files referenced by file-name-1 in another USE statement specifying the same condition.

14. After execution of the USE procedure, control is transferred to the invoking routine in the input-output control system. If the I-O status value does not indicate a critical input-output error, the input-output control system returns control to the next executable statement following the input-output statement whose execution caused the exception. If the I-O status value does indicate a critical error, the status is set normally, but no extraordinary action occurs. (See I-O status.)

WAIT Statement

WAIT is used for thread synchronization, allowing one thread to wait on another signal thread.

General format

WAIT FOR {{THREAD {object-reference-1 | thread-handle-1}} | {LAST THREAD} | {ANY THREAD}}
[{BEFORE TIME literal-1 | numeric-identifier-1}
{WITH TEST ONLY}
{WITH NO WAIT}
{THREAD IN thread-handle-2}
{SIZE OF identifier-2}
{STATUS IN identifier-3}]…
[ON EXCEPTION imperative-statement-1]
[NOT ON EXCEPTION imperative-statement-2]
[END-WAIT]

Syntax rules

1. WITH TEST ONLY and WITH NO WAIT are equivalent to BEFORE TIME 0.
2. Thread-handle-1 and thread-handle-2 must be USAGE HANDLE OF THREAD data items.
3. Identifier-3 must be a two character group item.

General rules

1. WAIT waits for another, the signal thread, to terminate or SEND a message.
2. The signal thread is the THREAD thread-handle-1 thread, the LAST thread, or for ANY THREAD, the first signal thread to terminate or send an appropriate message.
3. WAIT finishes immediately if there is already a message waiting for it.
4. A timeout in hundredths of a second may be specified by using BEFORE TIME, WITH TEST ONLY, or WITH NO WAIT. A timeout of 0 is effectively only tests the message status and returns. WITH TEST ONLY and WITH NO WAIT are therefore the same as BEFORE TIME 0.
5. The signal thread is stored in THREAD IN thread-handle-2.
6. The size of the message is stored in identifier-2.
7. The status codes are from the following:

00	Success
10	Thread Inactive Exception
99	Timeout Exception

15. If an exception occurred, imperative-statement-2 is executed. The flow of control then continues after the END-WAIT unless imperative-statement-2 transferred the flow of control.
16. If no exception occurred, imperative-statement-1 is executed. The flow of control then continues after the END-WAIT unless imperative-statement-1 transferred the flow of control.
17. SEND/RECEIVE/WAIT work across sessions as well as threads and may be used for inter-session communication on the same computer.
18. See SEND/RECEIVE.

WRITE Statement

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

The WRITE statement is also used to control DDS files, including display files. (DDS)

General format

Format 1 (sequential)

WRITE record-name-1 [FROM identifier-1 | literal-1]

[{BEFORE | AFTER} ADVANCING { PAGE | {{identifier-2 | integer-1} [LINE | LINES]}}]

[WITH HOLD]

[WITH NO LOCK]

[WITH KEPT LOCK]

[WITH WAIT]

[WITH IGNORE LOCK]

[ADVANCING ON LOCK]

[IGNORING LOCK]

[ALLOWING UPDATERS]

[WITH NO CONTROL]

[retry-phrase]

[NULL-MAP null-name]

[AT {END-OF-PAGE | EOP} imperative-statement-1]

[NOT AT {END-OF-PAGE | EOP} imperative-statement-2]

[END-WRITE]

Format 2 (random)

WRITE record-name-1 [FROM identifier-1 | literal-1]

[WITH HOLD]

[WITH NO LOCK]

[WITH KEPT LOCK]

[WITH WAIT]

[WITH IGNORE LOCK]

[ADVANCING ON LOCK]

[IGNORING LOCK]

[ALLOWING UPDATERS]

[WITH NO CONTROL]

[retry-phrase]

[FORMAT format-name]

[NULL-KEY-MAP null-key-name]

[NULL-MAP null-name]

[INVALID KEY imperative-statement-1]

[NOT INVALID KEY imperative-statement-2]

[END-WRITE]

Format 3 (transaction subfile DDS)

WRITE SUBFILE record-name-1 [FROM identifier-1 | literal-1]

[FORMAT format-name]

[TERMINAL terminal-name]

[INDICATOR indicator-name]

[INVALID KEY imperative-statement-1]

[NOT INVALID KEY imperative-statement-2]

[END-WRITE]

Format 4 (transaction non-subfile DDS)

WRITE record-name-1 [FROM identifier-1 | literal-1]

[{BEFORE | AFTER} ROLLING [LINE | LINES] {identifier-3 | literal-2} [THROUGH | THRU] {identifier-4 | literal-3} { UP | DOWN } {identifier-5 | literal-4} [LINE | LINES]]

[FORMAT format-name]

[TERMINAL terminal-name]

[{INDICATOR | INDICATORS | INDIC} indicator-name]

[STARTING AT LINE starting-line]

[INVALID KEY imperative-statement-1]

[NOT INVALID KEY imperative-statement-2]

[END-WRITE]

where retry-phrase is defined in RETRY phrase.

Syntax rules

1. Format 1 shall be used for files with sequential organization.
2. Format 2 shall be used for mass storage files with other than sequential organization.
3. If identifier-1 is a function-identifier, it shall reference an alphanumeric or national function. Record-name-1 and identifier-1 shall not reference the same storage area.
4. Record-name-1 is the name of a logical record in the file section of the data division and may be qualified.
5. If record-name-1 is specified, identifier-1 or literal-1 shall be valid as a sending operand in a MOVE statement specifying record-name-1 as the receiving operand.
6. If file-name-1 is specified, literal-1 shall be an alphanumeric, Boolean, or national literal. A figurative constant shall not be specified.
7. File-name-1 shall not reference a report file or a sort-merge file description entry.
8. Identifier-2 shall reference an integer data item.
9. Integer-1 may be positive or zero, but shall not be negative.
10. The phrases ADVANCING PAGE and END-OF-PAGE shall not both be specified in a single WRITE statement.
11. If the END-OF-PAGE or the NOT END-OF-PAGE phrase is specified, the LINAGE clause shall be specified in the file description entry for the associated file.
12. The words END-OF-PAGE and EOP are equivalent.
13. If record-name-1 is defined in a containing program, the file description entry for the file associated with record-name-1 shall contain a GLOBAL clause.

Syntax rules (DDS)

1. Format-name is a nonnumeric identifier or literal; it must be uppercase and at most 10 characters.

19. Terminal-name is a nonnumeric identifier or literal.
20. Indicator-name is a numeric, numeric table, or group item.

21. Null-key-name is an identifier.
22. Null-name is an identifier.
23. Starting-line is an integer or integer identifier.

General rules

1. The file referenced by file-name-1 or the file-name associated with record-name-1 shall be open in the output, I-O, or extend mode at the time of the execution of this statement.
2. The logical record released by the successful execution of the WRITE statement is no longer available in the record area unless the file-name associated with record-name-1 is specified in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as the file associated with record-name-1 .
3. The result of the execution of a WRITE statement specifying record-name-1 and the FROM phrase is equivalent to the execution of the following statements in the order specified:
   a. The statement:

      MOVE identifier-1 TO record-name-1

      or

      MOVE literal-1 TO record-name-1

      according to the rules specified for the MOVE statement.
   b. The same WRITE statement without the FROM phrase.
4. The figurative constant SPACE when specified in the WRITE statement references one alphanumeric space character.
5. The result of execution of a WRITE statement with the FILE phrase is equivalent to the execution of the following in the order specified:
   a. The statement:

      MOVE identifier-1 TO implicit-record-1

      or

      MOVE literal-1 TO implicit-record-1
   b. The statement:

      WRITE implicit-record-1

   where implicit-record-1 refers to the record area for file-name-1 and is treated:
   a. when identifier-1 is a function-identifier, as though implicit-record-1 were a record description entry subordinate to the file description entry having the same category, the same length, and the same representation as the returned value of the function.
   b. When identifier-1 is not a function-identifier, as though implicit-record-1 were a record description entry subordinate to the file description entry having the same description as identifier-1, or

c. when literal-1 is specified, as though implicit-record-1 were a record description entry subordinate to the file description entry having the same category, the same length, and the same representation as literal-1.

6. After the execution of the WRITE statement is complete, the information in the area referenced by identifier-1 is available, even though the information in the area referenced by record-name-1 is not available except as specified by the SAME RECORD AREA clause.
7. If single record locking was specified for the file connector associated with file-name-1 or the file-name associated with record-name-1, any record locks associated with that file connector are released at the completion of the successful execution of the WRITE statement.
8. If the LOCK phrase is specified and record locks have an effect for the file connector referenced by file-name-1 or the file-name associated with record-name-1, the record lock associated with the record accessed is set when the execution of the WRITE statement is successful.
9. The file position indicator is not affected by the execution of a WRITE statement.
10. The execution of the WRITE statement causes the value of the I-O status of file-name-1 or the file-name associated with record-name-1 to be updated. (See I-O status.)
11. The execution of the WRITE statement releases a logical record to the operating system.
12. When record-name-1 is specified, if the number of character positions to be written to the file is greater than the number of character positions in record-name-1, the content of the character positions that extend beyond the end of record-name-1 are undefined.
13. The number of character positions in literal-1, the data item referenced by identifier-1, or the record referenced by record-name-1 shall not be larger than the largest or smaller than the smallest number of character positions allowed by the RECORD IS VARYING clause associated with file-name-1 or the file-name associated with record-name-1. In either of these cases the execution of the WRITE statement is unsuccessful, the WRITE operation does not take place, the content of the record area is unaffected and the I-O status of file-name-1 or the file associated with record-name-1 is set to a value indicating the cause of the condition. (See I-O status.)
14. If, during the successful execution of a WRITE statement with the NOT END-OF-PAGE phrase, the end-of-page condition does not occur, control is transferred to imperative-statement-2 after execution of the input-output operation.
15. For format 1, if the execution of the WRITE statement is unsuccessful, the I-O status of file-name-1 or the file-name associated with record-name-1 is updated and control is transferred according to the rules of the USE statement following the execution of any USE AFTER EXCEPTION procedure applicable to the file-name associated with record-name-1.
16. For format 2, transfer of control following the successful or unsuccessful execution of the WRITE operation depends on the presence or absence of the INVALID KEY and NOT INVALID KEY phrases. (See Invalid key condition.)

17. The END-WRITE phrase delimits the scope of the WRITE statement. (See Scope of statements.)
18. The RETRY phrase is used to control the behavior of the WRITE statement for files opened for file sharing for the case where resources needed to write a record are locked by another run unit. The I-O status is set in accordance with the rules in RETRY phrase.

SEQUENTIAL FILES

1. The successor relationship of a sequential file is established by the order of execution of WRITE statements when the file is created. The relationship does not change except when records are added to the end of a file .
2. When a sequential file is open in the extend mode, the execution of the WRITE statement will add records to the end of the file as though the file were open in the output mode. If there are records in the file, the first record written after the execution of the OPEN statement with the EXTEND phrase is the successor of the last record in the file.
3. If two or more file connectors for a sequential file add records by sharing the file after opening it in extend mode, the records are in an undefined order.
4. When an attempt is made to write beyond the externally defined boundaries of a sequential file, an exception condition exists and the contents of the record area are unaffected. The following actions take place:
    a. The value of the I-O status of file-name-1 or the file-name associated with record-name-1 is set to a value indicating a boundary violation. (See I-O status.)
    b. If a USE AFTER STANDARD EXCEPTION declarative is explicitly or implicitly specified for file-name-1 or the file-name associated with record-name-1, that declarative procedure will then be executed.
    c. If a USE AFTER STANDARD EXCEPTION declarative is not explicitly or implicitly specified for file-name-1 or the file-name associated with record-name-1, the result is undefined.
5. If the end of reel/unit is recognized and the externally defined boundaries of the file have not been exceeded, the following operations are executed:
    a. The standard ending reel/unit label procedure.
    b. A reel/unit swap. The current volume pointer is updated to point to the next reel/unit existing for the file.
    c. The standard beginning reel/unit label procedure.
6. Both the ADVANCING phrase and the END-OF-PAGE phrase allow control of the vertical positioning of each line on a representation of a printed page. If the ADVANCING phrase is used, advancing is provided as follows:
    a. If integer-1 or the value of the data item referenced by identifier-2 is positive, the representation of the printed page is advanced the number of lines equal to that value.

b. If the value of the data item referenced by identifier-2 is negative, the results are undefined.

c. If integer-1 or the value of the data item referenced by identifier-2 is zero, no repositioning of the representation of the printed page is performed.

d. If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced according to general rule 24a, 24b, 24c, and 24d above.

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

f. If PAGE is specified and the LINAGE clause is specified in the associated file description entry, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page. The repositioning is to the first line that may be written on the next logical page as specified in the LINAGE clause.

g. If PAGE is specified and the LINAGE clause is not specified in the associated file description entry , the record is presented on the physical page before or after (depending on the phrase used) the device is repositioned to the next physical page. The repositioning to the next physical page is accomplished by the runtime. If physical page has no meaning in conjunction with a specific device, advancing will be provided by the runtime to act as if the user had specified BEFORE or AFTER (depending on the phrase used) ADVANCING 1 LINE.

7. If the logical end of the representation of the printed page is reached during the execution of a WRITE statement with the END-OF-PAGE phrase, imperative-statement-1 specified in the END-OF-PAGE phrase is executed. The logical end is specified in the LINAGE clause associated with record-name-1.

8. An end-of-page condition occurs when the execution of a given WRITE statement with the END-OF-PAGE phrase causes printing or spacing within the footing area of a page body. This occurs when the execution of such a WRITE statement causes the LINAGE-COUNTER to equal or exceed the value specified by integer-2 or the data item referenced by data-name-2 of the LINAGE clause if specified. In this case, the WRITE statement is executed and then imperative-statement-1 in the END-OF-PAGE phrase is executed.

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

   This occurs when a WRITE statement, if executed, would cause the LINAGE-COUNTER to exceed the value specified by integer-1 or the data item referenced by data-name-1 of the LINAGE clause. In this case, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the first line that can be written on the next logical page as specified in the LINAGE clause. Imperative-statement-1 in the END-OF-PAGE phrase, if specified, is executed after the record is written and the device has been repositioned.

   A page overflow condition occurs when the execution of a given WRITE statement would cause LINAGE-COUNTER to simultaneously exceed the value

of both integer-2 or the data item referenced by data-name-2 of the LINAGE clause and integer-1 or the data item referenced by data-name-1 of the LINAGE clause.

RELATIVE AND INDEXED FILES

When the invalid key condition is recognized, the execution of the WRITE statement is unsuccessful, the content of the record area is unaffected, and the I-O status of file-name-1 or the file-name associated with record-name-1 is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules for an invalid key condition. (See I-O status, and Invalid key condition.)

RELATIVE FILES

1. When a relative file is opened with the file connector in the output mode, records may be placed into the file through that file connector by one of the following:
    a. If the access mode is sequential, the WRITE statement causes a record to be released to the mass storage control system (MSCS). The first record has a relative record number of one, and subsequent records released have relative record numbers of 2, 3, 4, ... If the RELATIVE KEY clause is specified for file-name-1 or the file-name associated with record-name-1, the relative record number of the record being released is moved into the relative key data item by the mass storage control system (MSCS) during execution of the WRITE statement according to the rules for the MOVE statement.
    b. If the access mode is random or dynamic, prior to the execution of the WRITE statement the value of the relative key data item shall be initialized by the program with the relative record number to be associated with the record that is to be written. That record is then released to the mass storage control system (MSCS) by execution of the WRITE statement.
2. When a relative file is opened with the file connector in the extend mode, records are inserted into the file through that file connector. The first record released to the mass storage control system (MSCS) has a relative record number one greater than the highest relative record number existing in the file. Subsequent records released to the mass storage control system (MSCS) have consecutively higher relative record numbers. If the RELATIVE KEY clause is specified for file-name-1 or the file-name associated with record-name-1, the relative record number of the record being released is moved into the relative key data item by the mass storage control system (MSCS) during execution of the WRITE statement according to the rules for the MOVE statement.
3. If two or more file connectors for a relative file add records by sharing the file after opening it in extend mode, the relative key values returned are ascending, but not necessarily consecutive.
4. When a relative file is opened in the I-O mode and the access mode is random or dynamic, records are to be inserted in the associated file. Prior to the execution of the WRITE statement, the value of the relative key data item shall be initialized by the program with the relative record number to be associated with the record that is to be written. That record is then released to the mass storage control system (MSCS) when the program executed the WRITE statement.

5. The invalid key condition exists under the following circumstances:
    a. When the access mode is random or dynamic, and the relative key data item specifies a record that already exists in the file, regardless of any locks that are associated with the record, or
    b. When an attempt is made to write beyond the externally defined boundaries of the file, or
    c. When the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file.

INDEXED FILES

1. Execution of a WRITE statement causes the content of the record area to be released. The mass storage control system (MSCS) utilizes the contents of the record keys in such a way that subsequent access of the record may be made based upon any of these specified record keys.
2. The value of the prime record key shall be unique within the records in the file.
3. The data item specified as the prime record key shall be set by the program to the desired value prior to the execution of the WRITE statement.
4. If the file is open with the file connector in the sequential access mode, records shall be released to the mass storage control system (MSCS) through that file connector in ascending order of prime record key values according to the collating sequence of the file. When the file is open with the file connector in the extend mode, the first record released to the MSCS shall have a prime record key whose value is greater than the highest prime record key value existing in the file when it was opened through that file connector and each subsequent record released to the MSCS through that file connector shall have a prime record key whose value is greater than the highest prime record key value written by this file connector.
5. If two or more file connectors for an indexed file add records by sharing the file after opening it in extend mode, the order of alternate keys allowing duplicates is undefined.
6. If the file is open with the file connector in the random or dynamic access mode, records may be released to the mass storage control system (MSCS) through that file connector in any program-specified order.
7. When the ALTERNATE RECORD KEY clause is specified in the file control entry for an indexed file, the value of the alternate record key may be non-unique only if the DUPLICATES phrase is specified for that data item. In this case the mass storage control system (MSCS) provides storage of records such that when records are accessed sequentially, the order of retrieval of those records is the order in which they are released to the MSCS.
8. The invalid key condition exists under the following circumstances:
    a. When the file is open in the sequential access mode, and the file also is open in the output or extend mode, and the value of the prime record key is not greater than the value of the prime record key of the previous record, or

b. When the file is open in the output or I-O mode, and the value of the prime record key equals the value of the prime record key of a record already existing in the file, regardless of any record locks that are associated with the record, or

c. When the file is open in the output, extend, or I-O mode, and the value of an alternate record key for which duplicates are not allowed equals the value of the corresponding data item of a record already existing in the file, regardless of any record locks that are associated with the record, or

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

General rules (DDS)

FORMAT phrase (DDS)

The FORMAT phrase is required when there are multiple records for the given file. Format-name is the record format name the I-O operation uses to select which record format to use. Format-name must be an uppercase character string of at most 10 characters. A format of SPACES is treated as if there were no FORMAT phrase.

INDICATOR phrase (File DDS)

NULL-KEY-MAP phrase (File DDS)

The NULL-KEY-MAP phrase specifies the null-byte map value for DATABASE files with the ALWNULL attribute.

NULL-MAP (File DDS)

The NULL-MAP phrase specifies the null-byte map value for the DATABASE file with the ALWNULL attribute.

STARTING AT LINE phrase (Display DDS)

The STARTING phrase specifies the starting line number for record formats using the variable starting line keyword.

The actual line number used is the starting line plus DDS starting line minus one.

ROLLING phrase (Display DDS)

The ROLLING phrase allows display lines to be moved, up or down. All lines vacated are cleared.

14. Special Registers

Special Registers are not intrinsic functions, but are generally usable in the places where intrinsic functions are available. The concept of Special Registers precedes intrinsic functions and has been generally replaced by the intrinsic functions. These Special Registers, aside from the standard LINAGE-COUNTER, should generally be avoided in new code where possible.

Type Integer

LENGTH OF identifier
LINAGE-COUNTER OF file-id
LINAGE-COUNTER
RECORD-POSITION OF identifier
SIZE OF identifier
TIME-OF-DAY

Type Alphanumeric

CURRENT-DATE
CURRENT-TIME
FAC OF identifier
WHEN-COMPILED

CURRENT-DATE

The CURRENT-DATE special register returns the current date at time of execution in mm/dd/yy format. This function is inherently not Y2K safe because of the two-digit year. Use intrinsic functions instead.

General format

CURRENT-DATE

Sample Program

```
identification division.
program-id. testsr.

procedure division.
main-paragraph.

display "CURRENT-DATE =" current-date upon sysout
.
```

Sample Output

```
CURRENT-DATE =02/05/01
```

CURRENT-TIME

The CURRENT-TIME special register returns the current time at time of execution in hh:mm:ss format. Use intrinsic functions instead.

General format

CURRENT-TIME

Sample Program

```
identification division.
program-id. testsr.

procedure division.
main-paragraph.

display "CURRENT-TIME =" current-time upon sysout
.
```

Sample Output:

```
CURRENT-TIME =18:14:18
```

FAC

The FAC OF special register returns the Wang compatible FAC of a screen item. Do not use this special register in new code.

General format

FAC OF argument-1

Arguments

Argument-1 is a screen item.

LENGTH

The LENGTH OF special register returns the length of the given identifier in bytes. Use FUNCTION LENGTH instead.

General format

LENGTH OF argument-1

Arguments

Argument-1 is an alphanumeric or numeric class identifier.

Sample Program

```
identification division.
program-id. testsr.

data division.
working-storage section.

01 group-item.
   05 first-name pic x(10).
   05 filler pic x value " ".
   05 middle-initial pic x.
   05 filler pic x value " ".
   05 last-name pic x(20).
```

```
procedure division.
main-paragraph.

   move "George" to first-name
   move "W" to middle-initial
   move "Bush" to last-name

 display "group-item='" group-item "'" upon sysout
 display "" upon sysout

 display "LENGTH OF:" upon sysout
 display "group-item    =" length of group-item upon sysout
 display "first-name    =" length of first-name upon sysout
 display "middle-initial=" length of middle-initial upon sysout
 display "last-name     =" length of last-name upon sysout
 .
```

Sample Output:

```
group-item='George     W Bush               '

LENGTH OF:
group-item    =33
first-name    =10
middle-initial=1
last-name     =20
```

LINAGE-COUNTER

The LINAGE-COUNTER special register is used to refer to the implicit LINAGE-COUNTER of a file or the default LINAGE-COUNTER. The LINAGE-COUNTER is used to count the number of lines written in a sequential print file. The LINAGE-COUNTER may not be modified directly. It may be referenced only in procedure division statements; however, the input-output control system may change the value of LINAGE-COUNTER. The LINAGE-COUNTER is standard ANSI COBOL.

General Format

LINAGE-COUNTER [OF argument-1]

Arguments

Argument-1 is a file identifier.

Returned Values

The return value is the current LINAGE-COUNTER of the given file or the default LINAGE-COUNTER.

RECORD-POSITION

The RECORD-POSITION OF special register returns the offset of an identifier into the current record in bytes.

General format

RECORD-POSITION OF argument-1

Arguments

Argument-1 is an alphanumeric or numeric class identifier.

Sample Program

```
identification division.
program-id. testsr.

data division.
working-storage section.

01 group-item.
   05 first-name pic x(10).
   05 filler pic x value " ".
   05 middle-initial pic x.
   05 filler pic x value " ".
   05 last-name pic x(20).

procedure division.
main-paragraph.

move "George" to first-name
move "W" to middle-initial
move "Bush" to last-name

display "group-item='" group-item "'" upon sysout
display "" upon sysout

display "RECORD-POSITION OF:" upon sysout
display "group-item    =" record-position of group-item upon sysout
display "first-name    =" record-position of first-name upon sysout
display "middle-initial=" record-position of middle-initial upon sysout
display "last-name     =" record-position of last-name upon sysout
   .
```

Sample Output:

```
group-item='George     W Bush               '

RECORD-POSITION OF:
group-item    =0
first-name    =0
middle-initial=11
last-name     =13
```

LINAGE-COUNTER

The LINAGE-COUNTER special register is used to refer to the implicit LINAGE-COUNTER of a file or the default LINAGE-COUNTER. The LINAGE-COUNTER is used to count the number of lines written in a sequential print file. The LINAGE-COUNTER may not be modified directly. The LINAGE-COUNTER is standard ANSI COBOL.

General Format

LINAGE-COUNTER [OF argument-1]

Arguments

Argument-1 is a file identifier.

Returned Values

The return value is the current LINAGE-COUNTER of the given file or the default LINAGE-COUNTER.

SIZE

The SIZE OF special register returns the length of the given identifier in bytes, the same as the LENGTH special register or FUNCTION LENGTH. Use FUNCTION LENGTH instead.

General format

SIZE OF argument-1

Arguments

Argument-1 is an alphanumeric or numeric class identifier.

Sample Program

```
identification division.
program-id. testsr.

data division.
working-storage section.

01 group-item.
   05 first-name pic x(10).
   05 filler pic x value " ".
   05 middle-initial pic x.
   05 filler pic x value " ".
   05 last-name pic x(20).

procedure division.
main-paragraph.

   move "George" to first-name
   move "W" to middle-initial
   move "Bush" to last-name

   display "group-item='" group-item "'" upon sysout
   display "" upon sysout

   display "SIZE OF:" upon sysout
   display "group-item    =" size of group-item upon sysout
   display "first-name    =" size of first-name upon sysout
   display "middle-initial=" size of middle-initial upon sysout
   display "last-name     =" size of last-name upon sysout
   .
```

Sample Output (-comp:hp compiler option):

```
group-item='George     W Bush               '

SIZE OF:
group-item    =33
first-name    =10
middle-initial=1
last-name     =20
```

TIME-OF-DAY

The TIME-OF-DAY special register returns the current time of day while executing.

General format

TIME-OF-DAY

Sample Program

```
id division.
program-id. testsr.

data division.
working-storage section.

procedure division.
main-paragraph.

   display "TIME-OF-DAY  =" time-of-day upon sysout
   .
```

Sample Output:

```
TIME-OF-DAY  =181418
```

WHEN-COMPILED

The WHEN-COMPILED special register returns the date and time of compilation. The format of this special register is different with IBM and HP compatibility, so the -comp:hp flag has an effect. In the default IBM mode, the format is "mm/dd/yyhh.mm.ss". In HP mode, the format is "mm/dd/yy__hh:mm:ss'. Because of the two-digit years, neither mode of this special register is Y2K safe. Use the intrinsic FUNCTION WHEN-COMPILED instead of this special register.

General format

WHEN-COMPILED

Sample Program

```
id division.
program-id. testsr.

procedure division.
main-paragraph.

display "WHEN-COMPILED=" when-compiled upon sysout
```

.

Sample Output (standard compilation):

WHEN-COMPILED=02/05/0118.14.13

Sample Output (-comp:hp compiler option):

WHEN-COMPILED=02/05/01__18:14:34

15. Intrinsic functions

Each intrinsic function definition specifies:

1. Name and description of the function
2. Type of the function
3. General format of the function
4. Arguments, if any
5. Returned value

Refer to Function-identifier, for rules and explanations on the referencing of functions.

Types of Functions

Types of intrinsic functions are:

1. *Alphanumeric functions*. These are of the class and category alphanumeric. The number of character positions in this data item is specified in the function definition. Alphanumeric functions have an implicit usage display.
2. *National functions*. These are of the class and category national. The number of character positions in this data item is specified in the function definition. National functions have an implicit usage national.
3. *Numeric functions*. These are of the class and category numeric. A numeric function has an operational sign.
4. *Integer functions*. These are of the class and category numeric. An integer function has an operational sign and no digits to the right of the decimal point.

Arguments

Arguments specify values used in the evaluation of a function. Arguments are specified in the function-identifier. The definition of a function specifies the number of arguments required, which may be zero, one, or more. For some functions, the number of arguments may be variable. The order in which arguments are specified in a function-identifier determines the interpretation given to each value in arriving at the function value.

Arguments may be required to have a certain class or a subset of a certain class, to be a keyword, a type declaration, or a mnemonic-name. The types of argument are:

1. *Alphabetic*. An elementary data item of the class alphabetic or an alphanumeric literal containing only alphabetic characters shall be specified. The size associated with the argument may be used in determining the value of the function.

2. *Alphanumeric*. A data item of the class alphabetic or alphanumeric or an alphanumeric literal shall be specified. The size associated with the argument may be used in determining the value of the function.
3. *Index.* An index data item shall be specified. The size associated with the argument may be used in determining the value of the function.
4. *Integer*. An arithmetic expression that will always result in an integer value or an integer data item shall be specified. The value of the arithmetic expression, including operational sign, is used in determining the value of the function.
5. *Keyword*. A keyword shall be specified in accordance with the function definition.
6. *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.
7. *National.* An elementary data item of class national or a national literal shall be specified. The size associated with the argument may be used in determining the value of the function.
8. *Numeric.* An arithmetic expression or a numeric data item shall be specified. The value of the arithmetic expression, including operational sign, is used in determining the value of the function.
9. *Object.* An object identifier shall be specified. The size associated with the argument may be used in determining the value of the function.
10. *Pointer.* A pointer identifier shall be specified. The size associated with the argument may be used in determining the value of the function.

The rules for a function may place constraints on the permissible values for arguments in order to permit meaningful determination of the function's value.

When the definition of a function permits an argument to be repeated a variable number of times, a table may be referenced by specifying the data-name and any qualifiers that identify the table, followed immediately by subscripting where one or more of the subscripts is the word ALL.

When ALL is specified as a subscript, the effect is as if each table element associated with that subscript position were specified. The order of the implicit specification of each occurrence is from left to right, with the first (or leftmost) specification being the identifier with each subscript specified by the word ALL replaced by one, the next specification being the same identifier with the rightmost subscript specified by the word ALL incremented by one. The process continues with the rightmost ALL subscript being incremented by one for each implicit specification until the rightmost ALL subscript has been incremented through its range of values.

If there are any additional ALL subscripts, the ALL subscript immediately to the left of the rightmost ALL subscript is incremented by one, the rightmost ALL subscript is reset to one and the process of varying the rightmost ALL subscript is repeated. The ALL subscript to the left of the rightmost ALL subscript is incremented by one through its range of values.

For each additional ALL subscript, this process is repeated in turn until the leftmost ALL subscript has been incremented by one through its range of values. If the ALL

subscript is associated with a data item described with an OCCURS DEPENDING ON clause, the range of values is determined by the object of the OCCURS DEPENDING ON clause. The evaluation of an ALL subscript shall result in at least one argument; otherwise the result of the reference to the function-identifier is undefined.

Returned Values

The evaluation of a function produces a returned value in a temporary elementary data item. The type of a function identifies the type of the returned value as specified in Types of functions.

The returned value rules for certain integer and numeric intrinsic functions contain one or more equivalent arithmetic expressions. An equivalent arithmetic expression is a formal definition that defines the relationship among a function, its arguments, and its returned value. In the presentation of the equivalent arithmetic expressions where there is a variable number of occurrences of an argument, the rules may contain an equivalent arithmetic expression for one, two, and n occurrences.

The returned value of numeric and integer functions depends on the mode of arithmetic in effect, native or standard, and on whether an equivalent arithmetic expression is specified for the function.

Date Conversion Functions

The Gregorian calendar is used in the date conversion functions. The starting date of Monday, January 1, 1601, was chosen to establish a simple relationship between the Standard Date and DAY-OF-WEEK: integer date 1 was a Monday, DAY-OF-WEEK 1.

The method of calculating leap years is specified in ISO 8601:1988.

Summary of Functions

Table of functions, summarizes the functions that are available.

The "arguments" column defines argument type and the "type" column defines the type of the function, as follows:

Type	Meaning
Alph	Alphabetic
Anum	Alphanumeric
Ind	Index
Int	Integer
Key	Keyword
Mne	Mnemonic-name
Nat	National
Num	Numeric
Obj	Object
Ptr	Pointer

NOTE - Num in the arguments column includes Int. Both Int and Num are listed in the arguments column when the type of the argument determines the type of the function.

Table of functions

Intrinsic-function-name	Arguments	Type	Value returned
ABS	Int1 or Num1	Depends upon argument	The absolute value of argument
ACOS	Num1	Num	Arccosine of Num1
ALLOCATED-OCCURRENCES	Alph1 or Anum1 or Nat1 or Num1	Int	Number of occurrences currently allocated for a dynamic table
ANNUITY	Num1, Int2	Num	Ratio of annuity paid for Int2 periods at interest of Num1 to initial investment of one
ARGUMENT	Alph1 or Anum1 or Num1	Anum	Argument passed to program as parameter or property.
ARGUMENT-LENGTH	Num1	Num	Length of the command line in parameters.
ASIN	Num1	Num	Arcsine of Num1
ATAN	Num1	Num	Arctangent of Num1
ATAN2	Num1	Num	Arctangent of Num1
CHAR	Int1	Anum	Character in position Int1 of the alphanumeric program collating sequence
CHAR-NATIONAL	Int1	Nat	Character in position Int1 of the national program collating sequence
COS	Num1	Num	Cosine of Num1
COLUMN	Num1	Num	Column number currently being compiled.
CURRENT-DATE		Anum	Current date and time and difference from Greenwich Mean Time
DATE-OF-INTEGER	Int1	Int	Standard date equivalent (YYYYMMDD) of integer date
DATE-TO-YYYYMMDD	Int1, Int2	Int	Argument-1 converted from YYMMDD to YYYYMMDD based on the value of argument-2
DAY-OF-INTEGER	Int1	Int	Julian date equivalent (YYYYDDD) of integer date
DAY-TO-YYYYDDD	Int1, Int2	Int	Argument-1 converted from YYDDD to YYYYDDD based on the value of argument-2
DISPLAY-OF	Nat1, Anum1	Anum	Usage display representation of argument Nat1
E		Num	The value of e, the natural base
EXP	Num1	Num	e raised to the power Num1
EXP10	Num1	Num	10 raised to the power Num1
FACTORIAL	Int1	Int	factorial of Int1
FILE	Anum1	Anum	Filename currently being compiled
HIGHEST-ALGEBRAIC	Int1 or Num1 or Anum1	Int Num	Greatest algebraic value that may be represented in the argument
INTEGER	Num1	Int	The greatest integer not greater than Num1
INTEGER-OF-DATE	Int1	Int	Integer date equivalent of standard date (YYYYMMDD)
INTEGER-OF-DAY	Int1	Int	Integer date equivalent of Julian date (YYYYDDD)
INTEGER-PART	Num1	Int	Integer part of Num1
LENGTH	Alph1 or Anum1 or	Int	Length of argument in number of character positions or number of

Intrinsic-function-name	Arguments	Type	Value returned
	Ind1 or Nat1 or Num1		Boolean positions
LENGTH-AN	Alph1 or Anum1 or Ind1 or Nat1 or Num1	Int	Length of argument in number of alphanumeric character positions
LINE	Num1	Num	Line number currently being compiled
LOG	Num1	Num	Natural logarithm of Num1
LOG10	Num1	Num	Logarithm to base 10 of Num1
LOWER-CASE	Alph1 or Anum1 or Nat1	Depends Upon argument*	All letters in the argument are set to lowercase
LOWEST-ALGEBRAIC	Int1 or Num1 or Anum1	Int Num	Lowest algebraic value that may be represented in the argument.
MAX	Alph1 ... or Anum1 ... or Ind1 or Int1 .... or Nat1 ... or Num1 ...	Depends upon arguments*	Value of maximum argument
MEAN	Num1 ...	Num	Arithmetic mean of arguments
MEDIAN	Num1 ...	Num	Median of arguments
MIDRANGE	Num1 ...	Num	Mean of minimum and maximum arguments
MIN	Alph1 ... or Anum1 ... or Ind1 or Int1 ... or Nat1 ... or Num1 ...	Depends upon arguments*	Value of minimum argument
MOD	Int1, Int2	Int	Int1 modulo Int2
NATIONAL-OF	Anum1, Nat1	Nat	Usage national representation of argument Anum1
NUMVAL	Anum1 or Nat1	Num	Numeric value of simple numeric string
NUMVAL-C	Anum1, Anum2 or Nat1, Nat2	Num	Numeric value of numeric string with optional commas and currency sign
NUMVAL-F	Anum1 or Nat1	Num	Numeric value of numeric string representing a floating-point number
ORD	Alph1 or Anum1 or Nat1	Int	Ordinal position of the argument in collating sequence
ORD-MAX	Alph1 ... or Anum1 ... or Ind1 or Nat1 ... or Num1 ...	Int	Ordinal position of maximum argument
ORD-MIN	Alph1 ... or Anum1 ... or Ind1 or Nat1 ... or Num1 ...	Int	Ordinal position of minimum argument
PARAMETER	Num1	Num	Command-line parameter by number.
PI		Num	The value of p
PRESENT-VALUE	Num1, Num2 ...	Num	Present value of a series of future period-end amounts, Num2, at a

Intrinsic-function-name	Arguments	Type	Value returned
			discount rate of Num1
RANDOM	Int1	Num	Random number
RANGE	Int1 ... or Num1 ...	Depends upon argument	Value of maximum argument minus value of minimum argument
REM	Num1, Num2	Num	Remainder of Num1/Num2
REVERSE	Alph1 or Anum1 or Nat1	Depends upon argument*	Reverse order of the characters of the argument
SIGN	Num1	Int	The sign of Num1
SIN	Num1	Num	Sine of Num1
SOUNDEX	Alph1 or Anum1 or Nat1	Anum	Generate a lookup key when the proper spelling is not known for the query
SQRT	Num1	Num	Square root of Num1
STANDARD-COMPARE	Alph1, Anum1, or Nat1 Alph2, Anum2, or Nat2 Int3	Anum	A character indicating the result of comparing argument-1 to argument-2 using the ordering specified by ISO/IEC 14651 at the comparison level specified by argument-3
STANDARD-DEVIATION	Num1 ...	Num	Standard deviation of arguments
SUM	Int1 ... or Num1 ...	Depends upon arguments	Sum of arguments
TAN	Num1	Num	Tangent of Num1
UPPER-CASE	Alph1 or Anum1 or Nat1	Depends upon argument*	All letters in the argument are set to uppercase
URL-DECODE	Alph1 or Anum1 or Nat1	Anum	decodes a string encoded in URL format into text
URL-ENCODE	Alph1 or Anum1 or Nat1	Anum	encodes text into a string encoded URL format
VARIANCE	Num1 ...	Num	Variance of argument
WHEN-COMPILED		Anum	Date and time program was compiled
YEAR-TO-YYYY	Int1, Int2	Int	Argument-1 converted from YY to YYYY based on the value of argument-2
* A function that has only alphabetic arguments is type alphanumeric.			

ABS Function

The ABS function returns the absolute value of the argument.

The type of this function depends on the argument type as follows:

Argument type	Function type
Integer	Integer
Numeric	Numeric

General format

FUNCTION ABS (argument-1)

Arguments

Argument-1 shall be class numeric.

Returned values

The equivalent arithmetic expression shall be as follows:

1. When the value of argument-1 is zero or positive,
   (argument-1)
2. When the value of argument-1 is negative,
   (- argument-1)

Sample

```
display "FUNCTION ABS(numeric):" upon sysout
display FUNCTION ABS(987.654) upon sysout
display FUNCTION ABS(123) upon sysout
display FUNCTION ABS(-123) upon sysout
display FUNCTION ABS(-987.654) upon sysout
```

Sample Output

```
FUNCTION ABS(numeric):
987.654
123
123
987.654
```

ACOS Function

The ACOS function returns a numeric value in radians that approximates the arccosine of argument-1.

The type of this function is numeric.

General format

FUNCTION ACOS (argument-1)

Arguments

1. Argument-1 shall be class numeric.
2. The value of argument-1 shall be greater than or equal to -1 and less than or equal to +1.

Returned values

The returned value is the approximation of the arccosine of argument-1 and is greater than or equal to zero and less than or equal to pi.

Sample

```
display "FUNCTION ACOS(numeric):" upon sysout
display FUNCTION ACOS(1) upon sysout
display FUNCTION ACOS(.123) upon sysout
display FUNCTION ACOS(0) upon sysout
display FUNCTION ACOS(-.123) upon sysout
display FUNCTION ACOS(-1) upon sysout
```

Sample Output

```
FUNCTION ACOS(numeric):
0.0
```

1.4474840516030247
1.5707963267948966
1.6941086019867686
3.141592653589793

ADD-DURATION Function

The ADD-DURATION function adds a duration to a date, time or timestamp item, returning a new corresponding item.

General format

FUNCTION ADD-DURATION (argument-1 {date-time-keyword argument-3})

Where date-time-keyword is one of:

YEARS
MONTHS
DAYS
HOURS
MINUTES
SECONDS
MICROSECONDS

Arguments

Argument-1 is a date-time item.

Argument-3 is units of date-time-keyword. At least one date-time-keyword argument-3 pair must be specified, but each date-time-keyword may be specified no more than once.

Returned values

The date-time of argument-1 plus the duration specified.

Sample

```
 id division.
 program-id. add.

 data division.
  working-storage section.

  01 date-group.
 *                    "@Y-%m-%d"
     05 plain-date format of date.

 *                    "%H:%M:%S"
   01 time-group.
     05 plain-time format of time.

 *                    "@Y-%m-%d-%H.%M.%S.@Sm"
   01 timestamp-group.
     05 plain-timestamp format of timestamp.
      05 other-timestamp format of timestamp.

 procedure division.
 main-paragraph.
```

```
      move "1972-08-06" to plain-date
      move "12:34:56" to plain-time
      move "1972-08-06-12.34.56.123456" to plain-timestamp
      move "1972-09-08-16.11.22.333333" to other-timestamp
     .

ADD-DURATION.

  display "" upon sysout
  display "ADD-DURATION:" upon sysout
  display "" upon sysout

  display plain-date
          upon sysout

  assert plain-date = "1972-08-06"

  display function add-duration(plain-date years 1)
          upon sysout

  assert function add-duration(plain-date years 1)
         = "1973-08-06"

  display function add-duration(plain-date months 1)
          upon sysout

  assert function add-duration(plain-date months 1)
         = "1972-09-06"

  display function add-duration(plain-date days 1)
          upon sysout

  assert function add-duration(plain-date days 1)
         = "1972-08-07"

  display function add-duration(
           plain-date
           years 1
           months 1
           days 1
           )
          upon sysout

  assert function add-duration(
         plain-date years 1 months 1 days 1) = "1973-09-07"

  display plain-time
          upon sysout

  assert plain-time = "12:34:56"

      display function add-duration(plain-time hours 1)
          upon sysout

    assert function add-duration(plain-time hours 1) =
        "13:34:56"

   display function add-duration(plain-time minutes 1)
          upon sysout

    assert function add-duration(plain-time minutes 1) =
```

```
    "12:35:56"

 display function add-duration(plain-time seconds 1)
       upon sysout

  assert function add-duration(plain-time seconds 1) =
     "12:34:57"

 display function add-duration(
       plain-time
       hours 1
       minutes 1
       seconds 1
       )
       upon sysout

 assert function add-duration(plain-time hours 1
      minutes 1 seconds 1) = "13:35:57"

 display plain-timestamp
       upon sysout

 assert plain-timestamp = "1972-08-06-12.34.56.123456"

 display function add-duration(plain-timestamp years 1)
       upon sysout

 assert function add-duration(plain-timestamp years 1) =
     "1973-08-06-12.34.56.123456"

display function add-duration(plain-timestamp months 1)
       upon sysout

 assert function add-duration(plain-timestamp months 1) =
     "1972-09-06-12.34.56.123456"

display function add-duration(plain-timestamp days 1)
       upon sysout

 assert function add-duration(plain-timestamp days 1) =
     "1972-08-07-12.34.56.123456"

display function add-duration(plain-timestamp hours 1)
       upon sysout

 assert function add-duration(plain-timestamp hours 1) =
     "1972-08-06-13.34.56.123456"

display function add-duration(plain-timestamp minutes 1)
       upon sysout

 assert function add-duration(plain-timestamp minutes 1)=
     "1972-08-06-12.35.56.123456"

 display function add-duration(plain-timestamp seconds 1)
       upon sysout

 assert function add-duration(plain-timestamp seconds 1)=
     "1972-08-06-12.34.57.123456"

 display function add-duration(plain-timestamp microseconds 1)
       upon sysout
```

```
 assert function add-duration(plain-timestamp microseconds 1)=
    "1972-08-06-12.34.56.123457"

display function add-duration(
      plain-timestamp
      years 1
      months 1
      days 1
      hours 1
      minutes 1
      seconds 1
      microseconds 1
      )
     upon sysout

assert function add-duration(
      plain-timestamp
      years 1
      months 1
      days 1
      hours 1
      minutes 1
      seconds 1
      microseconds 1
      ) =
    "1973-09-07-13.35.57.123457"
        .
```

Sample Output

```
ADD-DURATION:

1972-08-06
1973-08-06
1972-09-06
1972-08-07
1973-09-07
12:34:56
13:34:56
12:35:56
12:34:57
13:35:57
1972-08-06-12.34.56.123456
1973-08-06-12.34.56.123456
1972-09-06-12.34.56.123456
1972-08-07-12.34.56.123456
1972-08-06-13.34.56.123456
1972-08-06-12.35.56.123456
1972-08-06-12.34.57.123456
1972-08-06-12.34.56.123457
1973-09-07-13.35.57.123457
```

ALLOCATED-OCCURRENCES Function

The ALLOCATED-OCCURRENCES function returns an integer that is the number of occurrences for which space is currently allocated for the table specified as the argument.

The type of this function is integer.

General Format

FUNCTION ALLOCATED-OCCURANCES (argument-1)

Arguments

Argument-1 shall be a table.

Returned Values

The returned value is the number of occurrences of the table specified in argument-1 for which storage is actually allocated.

Sample

```
identification division.
program-id. example.

data division.
working-storage section.

01 sample-table occurs 10 times.
  05 sample-item pic xx.

procedure division.
main-paragraph.

 display "FUNCTION ALLOCATED-OCCURANCES(table):" upon sysout
 display FUNCTION ALLOCATED-OCCURANCES(sample-table) upon sysout
```

Sample Output

```
FUNCTION ALLOCATED-OCCURANCES(table):
10
```

ANNUITY Function

The ANNUITY function (annuity immediate) returns a numeric value that approximates the ratio of an annuity paid at the end of each period for the number of periods specified by argument-2 to an initial investment of one. Interest is earned at the rate specified by argument-1 and is applied at the end of the period, before the payment.

The type of this function is numeric.

General format

FUNCTION ANNUITY (argument-1 argument-2)

Arguments

1. Argument-1 shall be class numeric.
2. The value of argument-1 shall be greater than or equal to zero.
3. Argument-2 shall be a positive integer.

Returned values

The equivalent arithmetic expression shall be as follows:

a. When the value of argument-1 is zero,
   (1 / argument-2)

b. When the value of argument-1 is not zero,
   (argument-1 / (1 - (1 + argument-1) (-argument-2)))**

Sample

```
display "FUNCTION ANNUITY(numeric,integer):" upon sysout
display FUNCTION ANNUITY(0, 4) upon sysout
display FUNCTION ANNUITY(2.9, 4) upon sysout
display FUNCTION ANNUITY(0, 2) upon sysout
```

Sample Output

```
FUNCTION ANNUITY(numeric,integer):
0.25
2.9125898601266536
0.5
```

ARGUMENT Function

The ARGUMENT function returns an alphanumeric value which represents the specified argument.

The type of this function is alphanumeric.

General format

FUNCTION ARGUMENT (argument-1)

Arguments

Argument-1 shall be class alphanumeric or numeric.

Returned values

The returned value is the contents represented by argument-1. Argument-1 may represent a command-line parameter, a system property, the special name “parameters” representing the number of command-line parameters passed, the special name “parameterxx” where xx is a two-digit number representing the parameter numbered xx.

Sample

```
java sample_args zero one two company=Heirloom four five product=COBOL

display "FUNCTION ARGUMENT(alphanumeric):" upon sysout
display FUNCTION ARGUMENT("company") upon sysout
display FUNCTION ARGUMENT("product") upon sysout
display FUNCTION ARGUMENT("parameter00") upon sysout
display FUNCTION ARGUMENT("parameter01") upon sysout
display FUNCTION ARGUMENT("parameter02") upon sysout
display FUNCTION ARGUMENT("parameter03") upon sysout
```

Sample Output

```
FUNCTION ARGUMENT(alphanumeric):
Heirloom
COBOL
zero
one
two
company=Heirloom
```

ARGUMENT-LENGTH Function

The ARGUMENT-LENGTH function returns the length of the command line in parameters.

General format

FUNCTION ARGUMENT-LENGTH

Returned values

The returned value is the number of parameters on the command line.

Sample Program

```
identification division.
program-id. sample_parameter.

procedure division.
main-paragraph.

 display "" upon sysout
 display "FUNCTION PARAMETER ( numeric ):" upon sysout
 display function parameter ( 0 ) upon sysout
 display function parameter ( 1 ) upon sysout
 display function argument-length upon sysout
 .
```

Sample Run

```
java sample_parameter alpha beta gamma delta
```

Sample Output

```
FUNCTION PARAMETER ( numeric ):
alpha
beta
4
```

ASIN Function

The ASIN function returns a numeric value in radians that approximates the arcsine of argument-1.

The type of this function is numeric.

General format

FUNCTION ASIN (argument-1)

Arguments

1. Argument-1 shall be class numeric.
2. The value of argument-1 shall be greater than or equal to B1 and less than or equal to +1.

Returned values

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.

Sample

```
display "FUNCTION ASIN(numeric):" upon sysout
display FUNCTION ASIN(1) upon sysout
display FUNCTION ASIN(.123) upon sysout
display FUNCTION ASIN(0) upon sysout
display FUNCTION ASIN(-.123) upon sysout
display FUNCTION ASIN(-1) upon sysout
```

Sample Output

```
FUNCTION ASIN(numeric):
1.5707963267948966
0.12331227519187199
0.0
-0.12331227519187199
-1.5707963267948966
```

ATAN Function

The ATAN function returns a numeric value in radians that approximates the arctangent of argument-1.

The type of this function is numeric.

General format

FUNCTION ATAN (argument-1)

Arguments

Argument-1 shall be class numeric.

Returned values

The returned value is the approximation of the arctangent of argument-1 and is greater than -pi/2 and less than +pi/2.

Sample

```
display "FUNCTION ATAN(numeric):" upon sysout
display FUNCTION ATAN(1000) upon sysout
display FUNCTION ATAN(100) upon sysout
display FUNCTION ATAN(1) upon sysout
display FUNCTION ATAN(.123) upon sysout
display FUNCTION ATAN(0) upon sysout
display FUNCTION ATAN(-.123) upon sysout
display FUNCTION ATAN(-1) upon sysout
display FUNCTION ATAN(-100) upon sysout
```

```
display FUNCTION ATAN(-1000) upon sysout
```

Sample Output

```
FUNCTION ATAN(numeric):
1.5697963271282298
1.5607966601082315
0.7853981633974483
0.12238528147180266
0.0
-0.12238528147180266
-0.7853981633974483
-1.5607966601082315
-1.5697963271282298
```

ATAN2 Function

The ATAN2 function converts rectangular coordinates (b,a) to polar coordinates (r,theta). This function uses both signs for computing the quadrant of the result.

General format

FUNCTION ATAN2 (argument-1 argument-2)

Arguments

Argument-1 and argument-2 shall be numeric.

Returned values

The returned value is the theta component of the point (r,theta) in polar coordinates that corresponds to the point (b,a) in Cartesian coordinates.

Sample Program

```
display "FUNCTION ATAN2(numeric numeric):" upon sysout
display FUNCTION ATAN2( .123 .456 ) upon sysout
display FUNCTION ATAN2( .234 .567 ) upon sysout
display FUNCTION ATAN2( .345 .678 ) upon sysout
display FUNCTION ATAN2( -.123 .456 ) upon sysout
display FUNCTION ATAN2( -.234 .567 ) upon sysout
display FUNCTION ATAN2( -.345 .678 ) upon sysout
display FUNCTION ATAN2( .123 -.456 ) upon sysout
display FUNCTION ATAN2( .234 -.567 ) upon sysout
display FUNCTION ATAN2( .345 -.678 ) upon sysout
display FUNCTION ATAN2( -.123 -.456 ) upon sysout
display FUNCTION ATAN2( -.234 -.567 ) upon sysout
display FUNCTION ATAN2( -.345 -.678 ) upon sysout
```

Sample Output

```
FUNCTION ATAN2(numeric numeric):
0.26346654103491746
0.39140512812521044
0.4707021656923164
-0.26346654103491746
-0.39140512812521044
-0.4707021656923164
```

2.8781261125548756
2.750187525464583
2.670890487897477
-2.8781261125548756
-2.750187525464583
-2.670890487897477

CHAR Function

The CHAR function returns a one-character alphanumeric value that is a character in the alphanumeric program collating sequence having the ordinal position equal to the value of argument-1.

The type of this function is alphanumeric.

General format

FUNCTION CHAR (argument-1)

Arguments

1. Argument-1 shall be an integer.
2. The value of argument-1 shall be greater than zero and less than or equal to the number of positions in the alphanumeric program collating sequence.

Returned values

1. The returned value shall be the character in the alphanumeric program collating sequence having the ordinal position specified by argument-1.
2. If more than one character has the same position in the alphanumeric 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.

Sample

```
display "FUNCTION CHAR(integer):" upon sysout
display FUNCTION CHAR(65) upon sysout
display FUNCTION CHAR(66) upon sysout
display FUNCTION CHAR(67) upon sysout
display FUNCTION CHAR(68) upon sysout
```

Sample Output

```
FUNCTION CHAR(integer):
@
A
B
C
```

CHAR-NATIONAL Function

The CHAR-NATIONAL (national character) function returns a one-character value that is a character in the national program collating sequence having the ordinal position equal to the value of the argument.

The type of the function is national.

General format

FUNCTION CHAR-NATIONAL (argument-1)

Arguments

1. Argument-1 shall be an integer.
2. The value of argument-1 shall be greater than zero and less than or equal to the number of positions in the national program collating sequence.

Returned values

1. The returned value shall be the character in the national program collating sequence having the ordinal position specified by argument-1.
2. If more than one character has the same ordinal position in the current national program collating sequence, the returned value shall be the first character defined for that position.

Sample

```
display "FUNCTION CHAR-NATIONAL(integer):" upon sysout
display FUNCTION CHAR-NATIONAL(65) upon sysout
display FUNCTION CHAR-NATIONAL(66) upon sysout
display FUNCTION CHAR-NATIONAL(67) upon sysout
display FUNCTION CHAR-NATIONAL(68) upon sysout
display FUNCTION CHAR-NATIONAL(256) upon sysout
display FUNCTION CHAR-NATIONAL(512) upon sysout
display FUNCTION CHAR-NATIONAL(1024) upon sysout
```

Sample Output

```
FUNCTION CHAR-NATIONAL(integer):
@
A
B
C
ÿ
?
?
```

COLUMN Function

The COLUMN function returns the column number currently being compiled to the nearest accuracy; it will actually return the column number containing the following token. It may be used for embedding messages about the source file itself within the executable. This function is resolved at compile-time, not run-time.

General format

FUNCTION COLUMN

Returned values

The returned value is the filename of the currently compiling file.

Sample Program

```
identification division.
program-id. sample_funcs.
```

```
procedure division.
main-paragraph.

  display "Compiling file '" function file "'" upon sysout

  display "Compiling line " function line "." upon sysout
  display "Compiling line " function line "." upon sysout
  display "Compiling line " function line "." upon sysout
  display "Compiling line " function line "." upon sysout
  display "Compiling line " function line "." upon sysout

  display "Compiling column " function column "." upon sysout
  display "Compiling column " function column "." upon sysout
  display "Compiling column " function column "." upon sysout
  display "Compiling column " function column "." upon sysout
  display "Compiling column " function column "." upon sysout
  .
```

Sample Output

```
Compiling file 'sample_funcs.cbl'
Compiling line 9.
Compiling line 10.
Compiling line 11.
Compiling line 12.
Compiling line 13.
Compiling column 49.
Compiling column 49.
Compiling column 49.
Compiling column 49.
Compiling column 49.
```

CONVERT-DATE-TIME Function

The CONVERT-DATE-TIME function converts an alphanumeric, numeric or date-time to a date-time.

General format

FUNCTION CONVERT-DATE-TIME (argument-1 date-time-type [LOCALE argument-3])

Where date-time-type is one of the following:

DATE

TIME

TIMESTAMP

Arguments

Argument-1 is a date, time or timestamp item, an item of class alphanumeric, a non-numeric literal, or an integer.

Argument-3 is a locale name.

Returned values

Sample

```
       id division.
       program-id. convert.

       data division.
       working-storage section.

         01 date-group.
       *                       "@Y-%m-%d"
            05 plain-date format of date.

       *                       "%H:%M:%S"
         01 time-group.
            05 plain-time format of time.

       *                       "@Y-%m-%d-%H.%M.%S.@Sm"
         01 timestamp-group.
            05 plain-timestamp format of timestamp.
              05 other-timestamp format of timestamp.

       procedure division.
       main-paragraph.

           move "1972-08-06" to plain-date
           move "12:34:56" to plain-time
           move "1972-08-06-12.34.56.123456" to plain-timestamp
           move "1972-09-08-16.11.22.333333" to other-timestamp
               .
       CONVERT-DATE-TIME-TEST.

         display "" upon sysout
         display "CONVERT-DATE-TIME (AS/400 strings):" upon sysout
         display "" upon sysout

         display "plain-timestamp=" plain-timestamp upon sysout

         display "@C=" function convert-date-time
                 ("0" timestamp "@C") upon sysout
         display "@p=" function convert-date-time
                 ("PM" timestamp "@p") upon sysout
         display "@Sh=" function convert-date-time
                 ("12" timestamp "@Sh") upon sysout
         display "@Sm=" function convert-date-time
                 ("123456" timestamp "@Sm") upon sysout
         display "@So=" function convert-date-time
                 ("123" timestamp "@So") upon sysout
         display "@St=" function convert-date-time
                 ("1" timestamp "@St") upon sysout
         display "@Y=" function convert-date-time
                 ("1972" timestamp "@Y") upon sysout

         display "%d=" function convert-date-time
                 ("06" timestamp "%d") upon sysout
         display "%D=" function convert-date-time
```

```
        ("08/06/72" timestamp "%D") upon sysout
    display "%H=" function convert-date-time
        ("12" timestamp "%H") upon sysout
    display "%I=" function convert-date-time
        ("12" timestamp "%I") upon sysout
    display "%j=" function convert-date-time
        ("219" timestamp "%j") upon sysout
    display "%m=" function convert-date-time
        ("08" timestamp "%m") upon sysout
    display "%M=" function convert-date-time
        ("34" timestamp "%M") upon sysout
    display "%p=" function convert-date-time
        ("PM" timestamp "%p") upon sysout
    display "%r=" function convert-date-time
        ("12:34:56 PM" timestamp "%r") upon sysout
    display "%R=" function convert-date-time
        ("12:34" timestamp "%R") upon sysout
    display "%S=" function convert-date-time
        ("56" timestamp "%S") upon sysout
    display "%y=" function convert-date-time
        ("72" timestamp "%y") upon sysout
    display "%Y=" function convert-date-time
        ("1972" timestamp "%Y") upon sysout
.

  *       #       INTU
  * &     INTF
  *       !       EXTF
  *       ^       EXTU
  *       $       NAML
  *       *       NAMS
  *       `       BINM
  *       ~       BINL

   CONVERT-DATE-TIME-INTF.

   display "" upon sysout
   display "CONVERT-DATE-TIME (internal format):" upon sysout
      display "" upon sysout

      display "&l=" function convert-date-time
          ("1" timestamp "&l") upon sysout
      display "&e=" function convert-date-time
          ("1" timestamp "&e") upon sysout
      display "&Y=" function convert-date-time
          ("1972" timestamp "&Y") upon sysout
      display "&y=" function convert-date-time
          ("72" timestamp "&y") upon sysout
      display "&c=" function convert-date-time
          ("19" timestamp "&c") upon sysout
      display "&C=" function convert-date-time
          ("0" timestamp "&C") upon sysout
      display "&N=" function convert-date-time
          ("1" timestamp "&N") upon sysout
      display "&m=" function convert-date-time
          ("07" timestamp "&m") upon sysout
```

```
    display "&u=" function convert-date-time
        ("33" timestamp "&u") upon sysout
    display "&W=" function convert-date-time
        ("2" timestamp "&W") upon sysout
    display "&d=" function convert-date-time
        ("06" timestamp "&d") upon sysout
    display "&j=" function convert-date-time
        ("219" timestamp "&j") upon sysout
    display "&w=" function convert-date-time
        ("1" timestamp "&w") upon sysout
    display "&U=" function convert-date-time
        ("1" timestamp "&U") upon sysout
    display "&A=" function convert-date-time
        ("1" timestamp "&A") upon sysout
    display "&I=" function convert-date-time
        ("12" timestamp "&I") upon sysout
    display "&H=" function convert-date-time
        ("12" timestamp "&H") upon sysout
    display "&M=" function convert-date-time
        ("34" timestamp "&M") upon sysout
    display "&S=" function convert-date-time
        ("56" timestamp "&S") upon sysout
    display "&s=" function convert-date-time
        ("123" timestamp "&s") upon sysout
    display "&z=" function convert-date-time
        ("-2880000" timestamp "&z") upon sysout
    display "&Z=" function convert-date-time
        ("03600000" timestamp "&Z") upon sysout
    display "&n=" function convert-date-time
        ("123456000" timestamp "&n") upon sysout
.

  CONVERT-DATE-TIME-EXTF.

   display "" upon sysout
    display "CONVERT-DATE-TIME (external format):" upon sysout
    display "" upon sysout

    display "!l=" function convert-date-time
        ("1" timestamp "!l") upon sysout
    display "!e=" function convert-date-time
        ("1" timestamp "!e") upon sysout
    display "!Y=" function convert-date-time
        ("1972" timestamp "!Y") upon sysout
    display "!y=" function convert-date-time
        ("72" timestamp "!y") upon sysout
    display "!c=" function convert-date-time
        ("19" timestamp "!c") upon sysout
    display "!C=" function convert-date-time
        ("0" timestamp "!C") upon sysout
    display "!N=" function convert-date-time
        ("1" timestamp "!N") upon sysout
    display "!m=" function convert-date-time
        ("08" timestamp "!m") upon sysout
    display "!u=" function convert-date-time
        ("33" timestamp "!u") upon sysout
```

```
    display "!W=" function convert-date-time
        ("2" timestamp "!W") upon sysout
    display "!d=" function convert-date-time
        ("06" timestamp "!d") upon sysout
    display "!j=" function convert-date-time
        ("219" timestamp "!j") upon sysout
    display "!w=" function convert-date-time
        ("1" timestamp "!w") upon sysout
    display "!U=" function convert-date-time
        ("1" timestamp "!U") upon sysout
    display "!A=" function convert-date-time
        ("1" timestamp "!A") upon sysout
    display "!I=" function convert-date-time
        ("12" timestamp "!I") upon sysout
    display "!H=" function convert-date-time
        ("12" timestamp "!H") upon sysout
    display "!M=" function convert-date-time
        ("34" timestamp "!M") upon sysout
    display "!S=" function convert-date-time
        ("56" timestamp "!S") upon sysout
    display "!s=" function convert-date-time
        ("123" timestamp "!s") upon sysout
    display "!z=" function convert-date-time
        ("-2880000" timestamp "!z") upon sysout
    display "!Z=" function convert-date-time
        ("03600000" timestamp "!Z") upon sysout
    display "!n=" function convert-date-time
        ("123456000" timestamp "!n") upon sysout
    .

CONVERT-DATE-TIME-NAML.

display "" upon sysout
  display "CONVERT-DATE-TIME (name long):" upon sysout
  display "" upon sysout

  display "$l=" function convert-date-time
        ("LEAP YEAR" timestamp "$l") upon sysout
  display "$e=" function convert-date-time
        ("CE" timestamp "$e") upon sysout
  display "$Y=" function convert-date-time
        ("1972" timestamp "$Y") upon sysout
  display "$y=" function convert-date-time
        ("72" timestamp "$y") upon sysout
  display "$c=" function convert-date-time
        ("19" timestamp "$c") upon sysout
  display "$C=" function convert-date-time
        ("0" timestamp "$C") upon sysout
  display "$N=" function convert-date-time
        ("1" timestamp "$N") upon sysout
  display "$m=" function convert-date-time
        ("August" timestamp "$m") upon sysout
  display "$u=" function convert-date-time
        ("33" timestamp "$u") upon sysout
  display "$W=" function convert-date-time
        ("2" timestamp "$W") upon sysout
```

```
    display "$d=" function convert-date-time
         ("06" timestamp "$d") upon sysout
    display "$j=" function convert-date-time
         ("219" timestamp "$j") upon sysout
    display "$w=" function convert-date-time
         ("Sunday" timestamp "$w") upon sysout
    display "$U=" function convert-date-time
         ("1" timestamp "$U") upon sysout
    display "$A=" function convert-date-time
         ("PM" timestamp "$A") upon sysout
    display "$I=" function convert-date-time
         ("12" timestamp "$I") upon sysout
    display "$H=" function convert-date-time
         ("12" timestamp "$H") upon sysout
    display "$M=" function convert-date-time
         ("34" timestamp "$M") upon sysout
    display "$S=" function convert-date-time
         ("56" timestamp "$S") upon sysout
    display "$s=" function convert-date-time
         ("123" timestamp "$s") upon sysout
    display "$z=" function convert-date-time
         ("-08:00" timestamp "$z") upon sysout
    display "$Z=" function convert-date-time
         ("+01:00" timestamp "$Z") upon sysout
    display "$n=" function convert-date-time
         ("123456000" timestamp "$n") upon sysout
    .

CONVERT-DATE-TIME-NAMS.

display "" upon sysout
    display "CONVERT-DATE-TIME (name short):" upon sysout
    display "" upon sysout

   display "*l=" function convert-date-time
         ("L" timestamp "*l") upon sysout
   display "*e=" function convert-date-time
         ("AD" timestamp "*e") upon sysout
   display "*Y=" function convert-date-time
         ("1972" timestamp "*Y") upon sysout
   display "*y=" function convert-date-time
         ("72" timestamp "*y") upon sysout
   display "*c=" function convert-date-time
         ("19" timestamp "*c") upon sysout
   display "*C=" function convert-date-time
         ("0" timestamp "*C") upon sysout
   display "*N=" function convert-date-time
         ("1" timestamp "*N") upon sysout
   display "*m=" function convert-date-time
         ("Aug" timestamp "*m") upon sysout
   display "*u=" function convert-date-time
         ("33" timestamp "*u") upon sysout
   display "*W=" function convert-date-time
         ("2" timestamp "*W") upon sysout
   display "*d=" function convert-date-time
         ("06" timestamp "*d") upon sysout
```

```
    display "*j=" function convert-date-time
        ("219" timestamp "*j") upon sysout
    display "*w=" function convert-date-time
        ("Sun" timestamp "*w") upon sysout
    display "*U=" function convert-date-time
        ("1" timestamp "*U") upon sysout
    display "*A=" function convert-date-time
        ("PM" timestamp "*A") upon sysout
    display "*I=" function convert-date-time
        ("12" timestamp "*I") upon sysout
    display "*H=" function convert-date-time
        ("12" timestamp "*H") upon sysout
    display "*M=" function convert-date-time
        ("34" timestamp "*M") upon sysout
    display "*S=" function convert-date-time
        ("56" timestamp "*S") upon sysout
    display "*s=" function convert-date-time
        ("123" timestamp "*s") upon sysout
    display "*z=" function convert-date-time
        ("-08:00" timestamp "*z") upon sysout
    display "*Z=" function convert-date-time
        ("+01:00" timestamp "*Z") upon sysout
    display "*n=" function convert-date-time
        ("123456000" timestamp "*n") upon sysout
.
```

Sample Output

CONVERT-DATE-TIME (AS/400 strings):

```
plain-timestamp=1972-08-06-12.34.56.123456
@C=2001-05-25 17:08:24.22
@p=2001-05-25 17:08:24.23
@Sh=2001-05-25 17:08:24.12
@Sm=2001-05-25 17:08:24.123456
@So=2001-05-25 17:08:24.123
@St=2001-05-25 17:08:24.1
@Y=1972-05-25 17:08:24.24
%d=2001-05-06 17:08:24.24
%D=1972-08-06 17:08:24.24
%H=2001-05-25 12:08:24.24
%I=2001-05-26 00:08:24.25
%j=2001-08-07 17:08:24.25
%m=2001-08-25 17:08:24.26
%M=2001-05-25 17:34:24.27
%p=2001-05-25 17:08:24.27
%r=2001-05-25 12:34:56.27
%R=2001-05-25 12:34:24.27
%S=2001-05-25 17:08:56.27
%y=1972-05-25 17:08:24.28
%Y=1972-05-25 17:08:24.28
```

CONVERT-DATE-TIME (internal format):

```
&I=2001-05-25 17:08:24.28
&e=2001-05-25 17:08:24.28
&Y=1972-05-25 17:08:24.29
&y=1972-05-25 17:08:24.29
&c=2001-05-25 17:08:24.33
```

&C=2001-05-25 17:08:24.33
&N=2001-05-25 17:08:24.33
&m=2001-08-25 17:08:24.33
&u=2001-08-17 17:08:24.33
&W=2001-05-11 17:08:24.33
&d=2001-05-06 17:08:24.34
&j=2001-08-07 17:08:24.34
&w=2001-05-20 17:08:24.36
&U=2001-05-04 17:08:24.36
&A=2001-05-25 17:08:24.36
&I=2001-05-26 00:08:24.36
&H=2001-05-25 12:08:24.36
&M=2001-05-25 17:34:24.37
&S=2001-05-25 17:08:56.37
&s=2001-05-25 17:08:24.123
&z=2001-05-25 16:40:24.38
&Z=2001-05-25 17:08:24.38
&n=2001-05-25 17:08:24.123456

CONVERT-DATE-TIME (external format):

!l=2001-05-25 17:08:24.41
!e=2001-05-25 17:08:24.41
!Y=1972-05-25 17:08:24.41
!y=1972-05-25 17:08:24.42
!c=2001-05-25 17:08:24.42
!C=2001-05-25 17:08:24.42
!N=2001-05-25 17:08:24.42
!m=2001-08-25 17:08:24.431
!u=2001-08-17 17:08:24.431
!W=2001-05-11 17:08:24.431
!d=2001-05-06 17:08:24.431
!j=2001-08-07 17:08:24.571
!w=2001-05-20 17:08:24.571
!U=2001-05-04 17:08:24.581
!A=2001-05-25 17:08:24.581
!I=2001-05-26 00:08:24.581
!H=2001-05-25 12:08:24.581
!M=2001-05-25 17:34:24.581
!S=2001-05-25 17:08:56.581
!s=2001-05-25 17:08:24.123
!z=2001-05-25 16:40:24.581
!Z=2001-05-25 17:08:24.591
!n=2001-05-25 17:08:24.123456

CONVERT-DATE-TIME (name long):

$l=2001-05-25 17:08:24.591
$e=2001-05-25 17:08:24.591
$Y=1972-05-25 17:08:24.591
$y=1972-05-25 17:08:24.591
$c=2001-05-25 17:08:24.591
$C=2001-05-25 17:08:24.601
$N=2001-05-25 17:08:24.601
$m=2001-08-25 17:08:24.601
$u=2001-08-17 17:08:24.601
$W=2001-05-11 17:08:24.601
$d=2001-05-06 17:08:24.611
$j=2001-08-07 17:08:24.611

$w=2001-05-26 17:08:24.611
$U=2001-05-04 17:08:24.611
$A=2001-05-25 17:08:24.611
$I=2001-05-26 00:08:24.611
$H=2001-05-25 12:08:24.611
$M=2001-05-25 17:34:24.611
$S=2001-05-25 17:08:56.621
$s=2001-05-25 17:08:24.123
$z=2001-05-25 17:08:24.621
$Z=2001-05-25 17:08:24.621
$n=2001-05-25 17:08:24.123456

CONVERT-DATE-TIME (name short):

*l=2001-05-25 17:08:24.631
*e=2001-05-25 17:08:24.631
*Y=1972-05-25 17:08:24.631
*y=1972-05-25 17:08:24.631
*c=2001-05-25 17:08:24.641
*C=2001-05-25 17:08:24.641
*N=2001-05-25 17:08:24.641
*m=2001-08-25 17:08:24.641
*u=2001-08-17 17:08:24.641
*W=2001-05-11 17:08:24.641
*d=2001-05-06 17:08:24.641
*j=2001-08-07 17:08:24.651
*w=2001-05-26 17:08:24.651
*U=2001-05-04 17:08:24.651
*A=2001-05-25 17:08:24.651
*I=2001-05-26 00:08:24.651
*H=2001-05-25 12:08:24.651
*M=2001-05-25 17:34:24.651
*S=2001-05-25 17:08:56.661
*s=2001-05-25 17:08:24.123
*z=2001-05-25 17:08:24.661
*Z=2001-05-25 17:08:24.661
*n=2001-05-25 17:08:24.123456

COS Function

The COS function returns a numeric value that approximates the cosine of an angle or arc, expressed in radians, that is specified by argument-1.

The type of this function is numeric.

General format

FUNCTION COS (argument-1)

Arguments

Argument-1 shall be class numeric.

Returned values

The returned value is the approximation of the cosine of argument-1 and is greater than or equal to -1 and less than or equal to +1.

Sample

```
display "FUNCTION COS(numeric):" upon sysout
display FUNCTION COS(1000) upon sysout
display FUNCTION COS(100) upon sysout
display FUNCTION COS(1) upon sysout
display FUNCTION COS(.123) upon sysout
display FUNCTION COS(0) upon sysout
display FUNCTION COS(-.123) upon sysout
display FUNCTION COS(-1) upon sysout
display FUNCTION COS(-100) upon sysout
display FUNCTION COS(-1000) upon sysout
```

Sample Output

```
FUNCTION COS(numeric):
0.5623790762907029
0.8623188722876839
0.5403023058681398
0.9924450321351935
1.0
0.9924450321351935
0.5403023058681398
0.8623188722876839
0.5623790762907029
```

CURRENT-DATE Function

The CURRENT-DATE function returns a 21-character alphanumeric value that represents the calendar date, time of day, and local time differential factor provided by the system on which the function is evaluated.

The type of this function is alphanumeric.

General format

FUNCTION CURRENT-DATE

Returned values

The character positions returned, numbered from left to right, are:

Character Positions	Contents
1-4	Four numeric digits of the year in the Gregorian calendar.
5-6	Two numeric digits of the month of the year, in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range 00 through 23.
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric digits of the seconds past the minute, in the range 00 through 59.
15-16	Two numeric digits of the hundredths of a second past the second, in the range 00 through 99. The value 00 is returned if the system on which the function is evaluated does not have the facility to provide the fractional part of a second.
17	Either the character '-', the character '+', or the character '0'. The character '-' is returned if the local time indicated in the previous character positions is behind Greenwich Mean Time. The character '+' is returned if the local time indicated is the same as or ahead of Greenwich Mean Time. The character '0' is returned if the system on which this function is evaluated does not have the facility to provide the local time differential factor.
18-19	If character position 17 is '-', two numeric digits are returned in the range 00 through

Character Positions	Contents
	12 indicating the number of hours that the reported time is behind Greenwich Mean Time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the reported time is ahead of Greenwich Mean Time. If character position 17 is '0', the value 00 is returned.
20-21	Two numeric digits are returned in the range 00 through 59 indicating the number of additional minutes that the reported time is ahead of or behind Greenwich Mean Time, depending on whether character position 17 is '+' or '-', respectively. If character position 17 is '0', the value 00 is returned.

If the system does not have the facility to provide fractional parts of a second, the value 00 is returned in character positions 15 and 16.

If the system does not have the facility to provide the local time differential factor, the value 00000 is returned in character positions 17 through 21.

Sample

```
display "FUNCTION CURRENT-DATE" upon sysout
display FUNCTION CURRENT-DATE upon sysout
```

Sample Output

```
FUNCTION CURRENT-DATE
2001020614211982-0800
```

DATE-OF-INTEGER Function

The DATE-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to standard date form (YYYYMMDD).

The type of this function is integer.

General format

FUNCTION DATE-OF-INTEGER (argument-1)

Arguments

Argument-1 is a positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar. It shall not exceed the value of FUNCTION INTEGER-OF-DATE(99991231).

Returned values

1. The returned value represents the ISO standard date equivalent of the integer specified in argument-1.
2. The returned value is in the form (YYYYMMDD) where YYYY represents a year in the Gregorian calendar; MM represents the month of that year; and DD represents the day of that month.

Sample

```
display "FUNCTION DATE-OF-INTEGER(numeric)" upon sysout
display FUNCTION DATE-OF-INTEGER(64103) upon sysout
display FUNCTION DATE-OF-INTEGER(134610) upon sysout
display FUNCTION DATE-OF-INTEGER(135484) upon sysout
display FUNCTION DATE-OF-INTEGER(135723) upon sysout
```

```
display FUNCTION DATE-OF-INTEGER(145732) upon sysout
display FUNCTION DATE-OF-INTEGER(146098) upon sysout
display FUNCTION DATE-OF-INTEGER(190841) upon sysout
```

Sample Output

```
FUNCTION DATE-OF-INTEGER(numeric)
17760704
19690720
19711211
```

DATE-TO-YYYYMMDD Function

The DATE-TO-YYYYMMDD function converts argument-1 from the form YYmmdd to the form YYYYmmdd. 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.

General format

FUNCTION DATE-TO-YYYYMMDD (argument-1 [argument-2])

Arguments

1. Argument-1 shall be zero or a positive integer less than 1000000.
2. Argument-2 shall be an integer.
3. If argument-2 is omitted, the function shall be evaluated as though 50 were specified.
4. The sum of the year at the time of execution and the value of argument-2 shall be less than 10000 and greater than 1699.

Returned values

The equivalent arithmetic expression shall be as follows:

(FUNCTION YEAR-TO-YYYY (YY, argument-2) * 10000 + mmdd)

where

YY = FUNCTION INTEGER (argument-1/10000)

mmdd = FUNCTION MOD (argument-1, 10000)

and where argument-1 of the INTEGER and MOD functions and argument-2 of the YEAR-TO-YYYY function are the same as argument-1 and argument-2 of the DATE-TO-YYYYMMDD function itself.

NOTES

1. 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.
2. This function supports a sliding window algorithm. See the notes for the YEAR-TO-YYYY function for a discussion of how to specify a fixed window.

Sample

```
display "FUNCTION DATE-TO-YYYYMMDD(YYMMDD):" upon sysout
display FUNCTION DATE-TO-YYYYMMDD(760704 50) upon sysout
display FUNCTION DATE-TO-YYYYMMDD(690720 50) upon sysout
display FUNCTION DATE-TO-YYYYMMDD(711211 50) upon sysout
display FUNCTION DATE-TO-YYYYMMDD(720806 50) upon sysout
display FUNCTION DATE-TO-YYYYMMDD(000101 50) upon sysout
display FUNCTION DATE-TO-YYYYMMDD(010101 50) upon sysout
display FUNCTION DATE-TO-YYYYMMDD(230704 50) upon sysout
```

Sample Output

```
FUNCTION DATE-TO-YYYYMMDD(YYMMDD):
19760704
19690720
19711211
19720806
20000101
20010101
20230704
```

3.

DAY-OF-INTEGER Function

The DAY-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to Julian date form (YYYYDDD).

The type of this function is integer.

General format

FUNCTION DAY-OF-INTEGER (argument-1)

Arguments

Argument-1 is a positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar. It shall not exceed the value of FUNCTION INTEGER-OF-DATE(99991231).

Returned values

1. The returned value represents the Julian equivalent of the integer specified in argument-1.
2. The returned value is an integer of the form (YYYYDDD) where YYYY represents a year in the Gregorian calendar and DDD represents the day of that year.

Sample

```
display "FUNCTION DAY-OF-INTEGER(numeric)" upon sysout
display FUNCTION DAY-OF-INTEGER(64103) upon sysout
display FUNCTION DAY-OF-INTEGER(134610) upon sysout
display FUNCTION DAY-OF-INTEGER(135484) upon sysout
display FUNCTION DAY-OF-INTEGER(135723) upon sysout
display FUNCTION DAY-OF-INTEGER(145732) upon sysout
display FUNCTION DAY-OF-INTEGER(146098) upon sysout
```

```
display FUNCTION DAY-OF-INTEGER(190841) upon sysout
```

Sample Output

```
FUNCTION DAY-OF-INTEGER(numeric)
1776186
1969201
1971345
1972219
2000001
2001001
2123185
```

DAY-TO-YYYYDDD Function

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.

General format

FUNCTION DAY-TO-YYYYMMDD (argument-1 [argument-2])

Arguments

1. Argument-1 shall be zero or a positive integer less than 100000.
2. Argument-2 shall be an integer.
3. If argument-2 is omitted, the function shall be evaluated as though 50 were specified.
4. The sum of the year at the time of execution and the value of argument-2 shall be less than 10000 and greater than 1699.

Returned values

The equivalent arithmetic expression shall be as follows:

(FUNCTION YEAR-TO-YYYY (YY, argument-2) * 1000 + nnn)

where

YY = FUNCTION INTEGER (argument-1/1000)

nnn = FUNCTION MOD (argument-1, 1000)

and where argument-1 of the INTEGER and MOD functions and argument-2 of the YEAR-TO-YYYY function are the same as argument-1 and argument-2 of the DAY-TO-YYYYDDD function itself.

NOTES

1. In the year 2002 the returned value for FUNCTION DAY-TO-YYYYDDD (10004, 20) is 2010004. In the year 2013 the returned value for FUNCTION DAY-TO-YYYYDDD (95005, (-10)) is 1995005.
2. This function supports a sliding window algorithm. See the notes for the YEAR-TO-YYYY function for a discussion of how to specify a fixed window.

Sample

```
display "FUNCTION DAY-TO-YYYYDDD(YYMMDD):" upon sysout
display FUNCTION DAY-TO-YYYYDDD(76050 50) upon sysout
display FUNCTION DAY-TO-YYYYDDD(69075 50) upon sysout
display FUNCTION DAY-TO-YYYYDDD(71100 50) upon sysout
display FUNCTION DAY-TO-YYYYDDD(72150 50) upon sysout
display FUNCTION DAY-TO-YYYYDDD(00200 50) upon sysout
display FUNCTION DAY-TO-YYYYDDD(01250 50) upon sysout
display FUNCTION DAY-TO-YYYYDDD(23300 50) upon sysout
```

Sample Output

```
FUNCTION DAY-TO-YYYYDDD(YYMMDD):
1976050
1969075
1971100
1972150
2000200
2001250
2023300
```

DISPLAY-OF Function

The DISPLAY-OF function returns a character string containing the external media format of the national characters in the argument.

The type of the function is alphanumeric.

General format

FUNCTION DISPLAY-OF (argument-1 [argument-2])

Arguments

1. Argument-1 shall be of class national.
2. Argument-2 shall be of class alphabetic or alphanumeric and shall be one character position in length. Argument-2 specifies an alphanumeric substitution character for use in conversion of national characters for which there is no corresponding alphanumeric character.

Returned values

1. A character string shall be returned with each national character of argument-1 converted to its corresponding alphanumeric character representation, if any. The correspondence of characters is supported between the alphanumeric character set and the national character set for purposes of conversion with the DISPLAY-OF function.
2. If argument-2 is specified, the alphanumeric substitution character shall be returned for each national character in argument-1 that has no corresponding alphanumeric character representation.
3. If argument-2 is unspecified, each national character in argument-1 that has no corresponding alphanumeric character representation shall be returned in national external media format, with the minimum of control functions necessary for interpretation of the external media format characters.

4. The length of the returned value shall be the number of character positions of usage display required to hold the returned value; it is dependent on the characters in the argument and on the attributes of the national character set.

Sample

```
identification division.
program-id. example.

data division.
working-storage section.

01 national-id pic n(10).

procedure division.
main-paragraph.
   display "FUNCTION DISPLAY-OF(national):" upon sysout
   move "Hello" to national-id
   display FUNCTION DISPLAY-OF ( national-id ) upon sysout
```

Sample Output

```
FUNCTION DISPLAY-OF(national):
Hello
```

E Function

The E function returns an approximation of e, the base of natural logarithms.

The type of the function is numeric.

General format

FUNCTION E

Returned values

The equivalent arithmetic expression shall be

(2 + .718281828459045235360287471352 6)

Sample

```
display "FUNCTION E:" upon sysout
display FUNCTION E upon sysout
```

Sample Output

```
FUNCTION E:
2.718281828459045
```

EXP Function

The EXP function returns an approximation of the value of e raised to the power of the argument.

The type of the function is numeric.

General format

FUNCTION EXP (argument-1)

Arguments

Argument-1 shall be class numeric.

Returned values

The equivalent arithmetic expression shall be:

(FUNCTION E ** argument-1)

Sample Output

```
FUNCTION EXP(numeric):
Infinity
2.6881171418161356E43
2.7182818284590455
1.1308844209474893
1.0
0.8842636625608209
0.36787944117144233
3.720075976020836E-44
0.0
```

EXP10 Function

The EXP10 function returns an approximation of the value of 10 raised to the power of the argument.

The type of the function is numeric.

General format

FUNCTION EXP10 (argument-1)

Arguments

Argument-1 shall be class numeric.

Returned values

The equivalent arithmetic expression shall be:

(10 ** argument-1)

Sample

```
display "FUNCTION EXP10(numeric):" upon sysout
display FUNCTION EXP10(1000) upon sysout
display FUNCTION EXP10(100) upon sysout
display FUNCTION EXP10(1) upon sysout
display FUNCTION EXP10(.123) upon sysout
display FUNCTION EXP10(0) upon sysout
display FUNCTION EXP10(-.123) upon sysout
display FUNCTION EXP10(-1) upon sysout
display FUNCTION EXP10(-100) upon sysout
display FUNCTION EXP10(-1000) upon sysout
```

Sample Output

```
FUNCTION EXP10(numeric):
Infinity
1.0E100
```

```
10.0
1.3273944577297394
1.0
0.7533555637337174
0.1
1.0E-100
0.0
```

EXTRACT-DATE-TIME Function

The EXTRACT-DATE-TIME function extracts a single attribute of a date-time item.

General format

FUNCTION EXTRACT-DATE-TIME (argument-1 {argument-2 | date-time-attribute})

Where date-time-attribute is one of the following:

YEARS
MONTHS
DAYS
HOURS
MINUTES
SECONDS
MICROSECONDS

Arguments

Argument-1 is a date, time or timestamp item.

If extract-type-keyword is used, the corresponding item is returned; if argument-2 is used, then argument-2 is a date-time format string, such as @Y for year.

Returned values

The extracted attribute of the date-time is returned, according to argument-2 format string or extract-type-keyword.

Sample

```
id division.
program-id. extract.

data division.
working-storage section.

  01 date-group.
*                    "@Y-%m-%d"
     05 plain-date format of date.

*                    "%H:%M:%S"
  01 time-group.
     05 plain-time format of time.

*                    "@Y-%m-%d-%H.%M.%S.@Sm"
  01 timestamp-group.
     05 plain-timestamp format of timestamp.
```

```
    05 other-timestamp format of timestamp.

procedure division.
main-paragraph.

      move "1972-08-06" to plain-date
      move "12:34:56" to plain-time
      move "1972-08-06-12.34.56.123456" to plain-timestamp
       move "1972-09-08-16.11.22.333333" to other-timestamp
       .

EXTRACT-DATE-TIME-KEYWORD.

  display "" upon sysout
  display "EXTRACT-DATE-TIME (keyword):" upon sysout
  display "" upon sysout

  display "plain-timestamp=" plain-timestamp upon sysout

  display function extract-date-time
        (plain-timestamp years) upon sysout
  assert function extract-date-time(plain-timestamp years)=
        "1972"

  display function extract-date-time
        (plain-timestamp months) upon sysout
  assert function extract-date-time(plain-timestamp months)=
        "8"

  display function extract-date-time
        (plain-timestamp days) upon sysout
  assert function extract-date-time(plain-timestamp days)=
        "6"

  display function extract-date-time
        (plain-timestamp hours) upon sysout
  assert function extract-date-time(plain-timestamp hours)=
        "12"

  display function extract-date-time
        (plain-timestamp minutes) upon sysout
  assert function extract-date-time(plain-timestamp minutes)=
        "34"

  display function extract-date-time
        (plain-timestamp seconds) upon sysout
  assert function extract-date-time(plain-timestamp seconds)=
        "56"

  display function extract-date-time
        (plain-timestamp microseconds) upon sysout
  assert function extract-date-time(plain-timestamp
        microseconds)="123456"
      .

EXTRACT-DATE-TIME-AS400.
```

```
display "" upon sysout
display "EXTRACT-DATE-TIME (AS/400 strings):" upon sysout
 display "" upon sysout

 display "plain-timestamp=" plain-timestamp upon sysout

 display "@C=" function extract-date-time
       (plain-timestamp "@C") upon sysout
 assert function extract-date-time
       (plain-timestamp "@C") ="0"
 display "@p=" function extract-date-time
       (plain-timestamp "@p") upon sysout
 display "@Sh=" function extract-date-time
       (plain-timestamp "@Sh") upon sysout
 display "@Sm=" function extract-date-time
       (plain-timestamp "@Sm") upon sysout
 display "@So=" function extract-date-time
       (plain-timestamp "@So") upon sysout
 display "@St=" function extract-date-time
       (plain-timestamp "@St") upon sysout
 display "@Y=" function extract-date-time
       (plain-timestamp "@Y") upon sysout

 display "%d=" function extract-date-time
       (plain-timestamp "%d") upon sysout
 display "%D=" function extract-date-time
       (plain-timestamp "%D") upon sysout
 display "%H=" function extract-date-time
       (plain-timestamp "%H") upon sysout
 display "%I=" function extract-date-time
       (plain-timestamp "%I") upon sysout
 display "%j=" function extract-date-time
       (plain-timestamp "%j") upon sysout
 display "%m=" function extract-date-time
       (plain-timestamp "%m") upon sysout
 display "%M=" function extract-date-time
       (plain-timestamp "%M") upon sysout
 display "%p=" function extract-date-time
       (plain-timestamp "%p") upon sysout
 display "%r=" function extract-date-time
       (plain-timestamp "%r") upon sysout
 display "%R=" function extract-date-time
       (plain-timestamp "%R") upon sysout
 display "%S=" function extract-date-time
       (plain-timestamp "%S") upon sysout
 display "%y=" function extract-date-time
       (plain-timestamp "%y") upon sysout
 display "%Y=" function extract-date-time
       (plain-timestamp "%Y") upon sysout

 display "%%=" function extract-date-time
       (plain-timestamp "%%") upon sysout
 display "@@=" function extract-date-time
       (plain-timestamp "@@") upon sysout
    .
```

```
 EXTRACT-DATE-TIME-INTU.

display "" upon sysout
  display "EXTRACT-DATE-TIME (internal unformat):" upon sysout
  display "" upon sysout

  display "#l=" function extract-date-time
        (plain-timestamp "#l") upon sysout
  display "#e=" function extract-date-time
        (plain-timestamp "#e") upon sysout
  display "#Y=" function extract-date-time
        (plain-timestamp "#Y") upon sysout
  display "#y=" function extract-date-time
        (plain-timestamp "#y") upon sysout
  display "#c=" function extract-date-time
        (plain-timestamp "#c") upon sysout
  display "#C=" function extract-date-time
        (plain-timestamp "#C") upon sysout
  display "#N=" function extract-date-time
        (plain-timestamp "#N") upon sysout
  display "#m=" function extract-date-time
        (plain-timestamp "#m") upon sysout
  display "#u=" function extract-date-time
        (plain-timestamp "#u") upon sysout
  display "#W=" function extract-date-time
        (plain-timestamp "#W") upon sysout
  display "#d=" function extract-date-time
        (plain-timestamp "#d") upon sysout
  display "#j=" function extract-date-time
        (plain-timestamp "#j") upon sysout
  display "#w=" function extract-date-time
        (plain-timestamp "#w") upon sysout
  display "#U=" function extract-date-time
        (plain-timestamp "#U") upon sysout
  display "#A=" function extract-date-time
        (plain-timestamp "#A") upon sysout
  display "#I=" function extract-date-time
        (plain-timestamp "#I") upon sysout
  display "#H=" function extract-date-time
        (plain-timestamp "#H") upon sysout
  display "#M=" function extract-date-time
        (plain-timestamp "#M") upon sysout
  display "#S=" function extract-date-time
        (plain-timestamp "#S") upon sysout
  display "#s=" function extract-date-time
        (plain-timestamp "#s") upon sysout
  display "#z=" function extract-date-time
        (plain-timestamp "#z") upon sysout
  display "#Z=" function extract-date-time
        (plain-timestamp "#Z") upon sysout
  display "#n=" function extract-date-time
        (plain-timestamp "#n") upon sysout
    .
 EXTRAct-DATE-TIME-INTF.
```

```
display "" upon sysout
 display "EXTRACT-DATE-TIME (internal format):" upon sysout
 display "" upon sysout

 display "&l=" function extract-date-time
       (plain-timestamp "&l") upon sysout
 display "&e=" function extract-date-time
       (plain-timestamp "&e") upon sysout
 display "&Y=" function extract-date-time
       (plain-timestamp "&Y") upon sysout
 display "&y=" function extract-date-time
       (plain-timestamp "&y") upon sysout
 display "&c=" function extract-date-time
       (plain-timestamp "&c") upon sysout
 display "&C=" function extract-date-time
       (plain-timestamp "&C") upon sysout
 display "&N=" function extract-date-time
       (plain-timestamp "&N") upon sysout
 display "&m=" function extract-date-time
       (plain-timestamp "&m") upon sysout
 display "&u=" function extract-date-time
       (plain-timestamp "&u") upon sysout
 display "&W=" function extract-date-time
       (plain-timestamp "&W") upon sysout
 display "&d=" function extract-date-time
       (plain-timestamp "&d") upon sysout
 display "&j=" function extract-date-time
       (plain-timestamp "&j") upon sysout
 display "&w=" function extract-date-time
       (plain-timestamp "&w") upon sysout
 display "&U=" function extract-date-time
       (plain-timestamp "&U") upon sysout
 display "&A=" function extract-date-time
       (plain-timestamp "&A") upon sysout
 display "&I=" function extract-date-time
       (plain-timestamp "&I") upon sysout
 display "&H=" function extract-date-time
       (plain-timestamp "&H") upon sysout
 display "&M=" function extract-date-time
       (plain-timestamp "&M") upon sysout
 display "&S=" function extract-date-time
       (plain-timestamp "&S") upon sysout
 display "&s=" function extract-date-time
       (plain-timestamp "&s") upon sysout
 display "&z=" function extract-date-time
       (plain-timestamp "&z") upon sysout
 display "&Z=" function extract-date-time
       (plain-timestamp "&Z") upon sysout
 display "&n=" function extract-date-time
       (plain-timestamp "&n") upon sysout
 .

EXTRACT-DATE-TIME-EXTU.

display "" upon sysout
 display "EXTRACT-DATE-TIME (external unformat):" upon sysout
```

```
 display "" upon sysout

 display "^l=" function extract-date-time
       (plain-timestamp "^l") upon sysout
 display "^e=" function extract-date-time
       (plain-timestamp "^e") upon sysout
 display "^Y=" function extract-date-time
       (plain-timestamp "^Y") upon sysout
 display "^y=" function extract-date-time
       (plain-timestamp "^y") upon sysout
 display "^c=" function extract-date-time
       (plain-timestamp "^c") upon sysout
 display "^C=" function extract-date-time
       (plain-timestamp "^C") upon sysout
 display "^N=" function extract-date-time
       (plain-timestamp "^N") upon sysout
 display "^m=" function extract-date-time
       (plain-timestamp "^m") upon sysout
 display "^u=" function extract-date-time
       (plain-timestamp "^u") upon sysout
 display "^W=" function extract-date-time
       (plain-timestamp "^W") upon sysout
 display "^d=" function extract-date-time
       (plain-timestamp "^d") upon sysout
 display "^j=" function extract-date-time
       (plain-timestamp "^j") upon sysout
 display "^w=" function extract-date-time
       (plain-timestamp "^w") upon sysout
 display "^U=" function extract-date-time
       (plain-timestamp "^U") upon sysout
 display "^A=" function extract-date-time
       (plain-timestamp "^A") upon sysout
 display "^I=" function extract-date-time
       (plain-timestamp "^I") upon sysout
 display "^H=" function extract-date-time
       (plain-timestamp "^H") upon sysout
 display "^M=" function extract-date-time
       (plain-timestamp "^M") upon sysout
 display "^S=" function extract-date-time
       (plain-timestamp "^S") upon sysout
 display "^s=" function extract-date-time
       (plain-timestamp "^s") upon sysout
 display "^z=" function extract-date-time
       (plain-timestamp "^z") upon sysout
 display "^Z=" function extract-date-time
       (plain-timestamp "^Z") upon sysout
 display "^n=" function extract-date-time
       (plain-timestamp "^n") upon sysout
 .

EXTRACT-DATE-TIME-EXTF.

display "" upon sysout
 display "EXTRACT-DATE-TIME (external format):" upon sysout
 display "" upon sysout
```

```
    display "!l=" function extract-date-time
          (plain-timestamp "!l") upon sysout
    display "!e=" function extract-date-time
          (plain-timestamp "!e") upon sysout
    display "!Y=" function extract-date-time
          (plain-timestamp "!Y") upon sysout
    display "!y=" function extract-date-time
          (plain-timestamp "!y") upon sysout
    display "!c=" function extract-date-time
          (plain-timestamp "!c") upon sysout
    display "!C=" function extract-date-time
          (plain-timestamp "!C") upon sysout
    display "!N=" function extract-date-time
          (plain-timestamp "!N") upon sysout
    display "!m=" function extract-date-time
          (plain-timestamp "!m") upon sysout
    display "!u=" function extract-date-time
          (plain-timestamp "!u") upon sysout
    display "!W=" function extract-date-time
          (plain-timestamp "!W") upon sysout
    display "!d=" function extract-date-time
          (plain-timestamp "!d") upon sysout
    display "!j=" function extract-date-time
          (plain-timestamp "!j") upon sysout
    display "!w=" function extract-date-time
          (plain-timestamp "!w") upon sysout
    display "!U=" function extract-date-time
          (plain-timestamp "!U") upon sysout
    display "!A=" function extract-date-time
          (plain-timestamp "!A") upon sysout
    display "!I=" function extract-date-time
          (plain-timestamp "!I") upon sysout
    display "!H=" function extract-date-time
          (plain-timestamp "!H") upon sysout
    display "!M=" function extract-date-time
          (plain-timestamp "!M") upon sysout
    display "!S=" function extract-date-time
          (plain-timestamp "!S") upon sysout
    display "!s=" function extract-date-time
          (plain-timestamp "!s") upon sysout
    display "!z=" function extract-date-time
          (plain-timestamp "!z") upon sysout
    display "!Z=" function extract-date-time
          (plain-timestamp "!Z") upon sysout
    display "!n=" function extract-date-time
          (plain-timestamp "!n") upon sysout
    .

  EXTRACT-DATE-TIME-NAML.

display "" upon sysout
    display "EXTRACT-DATE-TIME (name long):" upon sysout
    display "" upon sysout

    display "$l=" function extract-date-time
          (plain-timestamp "$l") upon sysout
```

```
    display "$e=" function extract-date-time
          (plain-timestamp "$e") upon sysout
    display "$Y=" function extract-date-time
          (plain-timestamp "$Y") upon sysout
    display "$y=" function extract-date-time
          (plain-timestamp "$y") upon sysout
    display "$c=" function extract-date-time
          (plain-timestamp "$c") upon sysout
    display "$C=" function extract-date-time
          (plain-timestamp "$C") upon sysout
    display "$N=" function extract-date-time
          (plain-timestamp "$N") upon sysout
    display "$m=" function extract-date-time
          (plain-timestamp "$m") upon sysout
    display "$u=" function extract-date-time
          (plain-timestamp "$u") upon sysout
    display "$W=" function extract-date-time
          (plain-timestamp "$W") upon sysout
    display "$d=" function extract-date-time
          (plain-timestamp "$d") upon sysout
    display "$j=" function extract-date-time
          (plain-timestamp "$j") upon sysout
    display "$w=" function extract-date-time
          (plain-timestamp "$w") upon sysout
    display "$U=" function extract-date-time
          (plain-timestamp "$U") upon sysout
    display "$A=" function extract-date-time
          (plain-timestamp "$A") upon sysout
    display "$I=" function extract-date-time
          (plain-timestamp "$I") upon sysout
    display "$H=" function extract-date-time
          (plain-timestamp "$H") upon sysout
    display "$M=" function extract-date-time
          (plain-timestamp "$M") upon sysout
    display "$S=" function extract-date-time
          (plain-timestamp "$S") upon sysout
    display "$s=" function extract-date-time
          (plain-timestamp "$s") upon sysout
    display "$z=" function extract-date-time
          (plain-timestamp "$z") upon sysout
    display "$Z=" function extract-date-time
          (plain-timestamp "$Z") upon sysout
    display "$n=" function extract-date-time
          (plain-timestamp "$n") upon sysout
    .

  EXTRACT-DATE-TIME-NAMS.

display "" upon sysout
    display "EXTRACT-DATE-TIME (name short):" upon sysout
    display "" upon sysout

    display "*l=" function extract-date-time
          (plain-timestamp "*l") upon sysout
    display "*e=" function extract-date-time
          (plain-timestamp "*e") upon sysout
```

```
    display "*Y=" function extract-date-time
        (plain-timestamp "*Y") upon sysout
    display "*y=" function extract-date-time
        (plain-timestamp "*y") upon sysout
    display "*c=" function extract-date-time
        (plain-timestamp "*c") upon sysout
    display "*C=" function extract-date-time
        (plain-timestamp "*C") upon sysout
    display "*N=" function extract-date-time
        (plain-timestamp "*N") upon sysout
    display "*m=" function extract-date-time
        (plain-timestamp "*m") upon sysout
    display "*u=" function extract-date-time
        (plain-timestamp "*u") upon sysout
    display "*W=" function extract-date-time
        (plain-timestamp "*W") upon sysout
    display "*d=" function extract-date-time
        (plain-timestamp "*d") upon sysout
    display "*j=" function extract-date-time
        (plain-timestamp "*j") upon sysout
    display "*w=" function extract-date-time
        (plain-timestamp "*w") upon sysout
    display "*U=" function extract-date-time
        (plain-timestamp "*U") upon sysout
    display "*A=" function extract-date-time
        (plain-timestamp "*A") upon sysout
    display "*I=" function extract-date-time
        (plain-timestamp "*I") upon sysout
    display "*H=" function extract-date-time
        (plain-timestamp "*H") upon sysout
    display "*M=" function extract-date-time
        (plain-timestamp "*M") upon sysout
    display "*S=" function extract-date-time
        (plain-timestamp "*S") upon sysout
    display "*s=" function extract-date-time
        (plain-timestamp "*s") upon sysout
    display "*z=" function extract-date-time
        (plain-timestamp "*z") upon sysout
    display "*Z=" function extract-date-time
        (plain-timestamp "*Z") upon sysout
    display "*n=" function extract-date-time
        (plain-timestamp "*n") upon sysout
      .
```

Sample Output

```
    EXTRACT-DATE-TIME (keyword):

    plain-timestamp=1972-08-06-12.34.56.123456
    1972
    8
    6
    12
    34
    56
    123456
```

EXTRACT-DATE-TIME (AS/400 strings):

plain-timestamp=1972-08-06-12.34.56.123456
@C=0
@p=PM
@Sh=12
@Sm=123456
@So=123
@St=1
@Y=1972
%d=06
%D=08/06/72
%H=12
%I=12
%j=219
%m=08
%M=34
%p=PM
%r=12:34:56 PM
%R=12:34
%S=56
%y=72
%Y=1972
%%=%
@@=@

EXTRACT-DATE-TIME (internal unformat):

#l=1
#e=1
#Y=1972
#y=72
#c=19
#C=0
#N=1
#m=7
#u=33
#W=2
#d=6
#j=219
#w=1
#U=1
#A=1
#I=12
#H=12
#M=34
#S=56
#s=123
#z=-28800000
#Z=3600000
#n=123456000

EXTRACT-DATE-TIME (internal format):

&l=1
&e=1

&Y=1972
&y=72
&c=19
&C=0
&N=1
&m=07
&u=33
&W=2
&d=06
&j=219
&w=1
&U=1
&A=1
&l=12
&H=12
&M=34
&S=56
&s=123
&z=-2880000
&Z=03600000
&n=123456000

EXTRACT-DATE-TIME (external unformat):

^l=1
^e=1
^Y=1972
^y=72
^c=19
^C=0
^N=1
^m=08
^u=33
^W=2
^d=06
^j=219
^w=1
^U=1
^A=1
^l=12
^H=12
^M=34
^S=56
^s=123
^z=-2880000
^Z=03600000
^n=123456000

EXTRACT-DATE-TIME (external format):

!l=1
!e=1
!Y=1972
!y=72
!c=19
!C=0

!N=1
!m=8
!u=33
!W=2
!d=6
!j=219
!w=1
!U=1
!A=1
!I=12
!H=12
!M=34
!S=56
!s=123
!z=-28800000
!Z=3600000
!n=123456000

EXTRACT-DATE-TIME (name long):

$l=LEAP YEAR
$e=CE
$Y=1972
$y=72
$c=19
$C=0
$N=1
$m=August
$u=33
$W=2
$d=06
$j=219
$w=Sunday
$U=1
$A=PM
$I=12
$H=12
$M=34
$S=56
$s=123
$z=-08:00
$Z=+01:00
$n=123456000

EXTRACT-DATE-TIME (name short):

*l=L
*e=AD
*Y=1972
*y=72
*c=19
*C=0
*N=1
*m=Aug
*u=33
*W=2

*d=06
*j=219
*w=Sun
*U=1
*A=PM
*I=12
*H=12
*M=34
*S=56
*s=123
*z=-08:00
*Z=+01:00
*n=123456000

FACTORIAL Function

The FACTORIAL function returns an integer that is the factorial of argument-1.

The type of this function is integer.

General format

FUNCTION FACTORIAL (argument-1)

Arguments

Argument-1 shall be an integer greater than or equal to zero.

Returned values

The equivalent arithmetic expression shall be as follows:

1. When the value of argument-1 is 0 or 1,
   (1)
2. When the value of argument-1 is 2,
   (2)
3. When the value of argument-1 is n,
   (n * (n - 1) * (n - 2) * ... * 1)

Sample

```
display "FUNCTION FACTORIAL(integer):" upon sysout
display FUNCTION FACTORIAL(0) upon sysout
display FUNCTION FACTORIAL(1) upon sysout
display FUNCTION FACTORIAL(2) upon sysout
display FUNCTION FACTORIAL(3) upon sysout
display FUNCTION FACTORIAL(4) upon sysout
display FUNCTION FACTORIAL(5) upon sysout
display FUNCTION FACTORIAL(6) upon sysout
display FUNCTION FACTORIAL(7) upon sysout
display FUNCTION FACTORIAL(8) upon sysout
display FUNCTION FACTORIAL(9) upon sysout
display FUNCTION FACTORIAL(10) upon sysout
```

Sample Output

```
FUNCTION FACTORIAL(integer):
```

```
1
1
2
6
24
120
720
5040
40320
362880
3628800
```

FILE Function

The FILE function returns the filename currently being compiled. It may be used for embedding messages about the source file itself within the executable. This function is resolved at compile-time, not run-time.

General format

FUNCTION FILE

Returned values

The returned value is the filename of the currently compiling file.

Sample Program

```
identification division.
program-id. sample_funcs.

procedure division.
main-paragraph.
   display "Compiling file '" function file "'" upon sysout

   display "Compiling line " function line "." upon sysout
   display "Compiling line " function line "." upon sysout
   display "Compiling line " function line "." upon sysout
   display "Compiling line " function line "." upon sysout
   display "Compiling line " function line "." upon sysout

   display "Compiling column " function column "." upon sysout
   display "Compiling column " function column "." upon sysout
   display "Compiling column " function column "." upon sysout
   display "Compiling column " function column "." upon sysout
   display "Compiling column " function column "." upon sysout
   .
```

Sample Output

```
Compiling file 'sample_funcs.cbl'
Compiling line 9.
Compiling line 10.
Compiling line 11.
Compiling line 12.
Compiling line 13.
Compiling column 49.
Compiling column 49.
```

Compiling column 49.
Compiling column 49.
Compiling column 49.

FIND-DURATION Function

The FIND-DURATION function calculates the duration of a particular date-time attribute between two date-time items.

General format

FUNCTION FIND-DURATION (argument-1 argument-2 date-time-attribute)

Where date-time-attribute is one of the following:

YEARS
MONTHS
DAYS
HOURS
MINUTES
SECONDS
MICROSECONDS

Arguments

Argument-1 and argument-2 must be date-time items.

Returned values

The difference in units of date-time-attribute between argument-1 and argument-2 is returned.

Sample

```
id division.
     program-id. find.

     data division.
     working-storage section.

     01 date-group.
    *                    "@Y-%m-%d"
        05 plain-date format of date.

    *                    "%H:%M:%S"
     01 time-group.
        05 plain-time format of time.

    *                    "@Y-%m-%d-%H.%M.%S.@Sm"
     01 timestamp-group.
        05 plain-timestamp format of timestamp.
      05 other-timestamp format of timestamp.

     procedure division.
     main-paragraph.

        move "1972-08-06" to plain-date
```

```
   move "12:34:56" to plain-time
   move "1972-08-06-12.34.56.123456" to plain-timestamp
 move "1972-09-08-16.11.22.333333" to other-timestamp
 .

FIND-DURATION.

   display "" upon sysout
 display "FIND-DURATION:" upon sysout
 display "" upon sysout

 display "arg1: " plain-timestamp upon sysout
 display "arg2: " other-timestamp upon sysout

 display "years: " function find-duration
    (
       plain-timestamp other-timestamp years
    )
    upon sysout
 display "months: " function find-duration
    (
       plain-timestamp other-timestamp months
    )
    upon sysout
 display "days: " function find-duration
    (
       plain-timestamp other-timestamp days
    )
    upon sysout
 display "hours: " function find-duration
    (
       plain-timestamp other-timestamp hours
    )
    upon sysout
 display "minutes: " function find-duration
    (
       plain-timestamp other-timestamp minutes
    )
    upon sysout
 display "seconds: " function find-duration
    (
       plain-timestamp other-timestamp seconds
    )
    upon sysout
 display "microseconds: " function find-duration
    (
       plain-timestamp other-timestamp microseconds
    )
    upon sysout
   .
```

Sample Output

FIND-DURATION:

arg1: 1972-08-06-12.34.56.123456
arg2: 1972-09-08-16.11.22.333333
years: 0
months: 1
days: 33
hours: 795
minutes: 47736
seconds: 2864186
microseconds: 2864186419877

HIGHEST-ALGEBRAIC Function

The HIGHEST-ALGEBRAIC function returns a value that is equal to the greatest algebraic value that may be represented in argument-1.

The type of this function depends upon the argument types as follows:

Argument type	Function type
Integer	Integer
Numeric	Numeric
Numeric-edited	Numeric

General format

FUNCTION HIGHEST-ALGEBRAIC (argument-1)

Arguments

Arguments-1 shall be an elementary data item of category numeric or numeric-edited.

Returned values

The value returned is equal to the positive algebraic value of greatest magnitude that may be represented in argument-1.

NOTE - The following illustrates the expected results for some values of argument-1.

Argument-1 characteristics	Value returned
S999	+999
S9(4) BINARY	+9999
99V9(3)	+99.999
$**,**9.99BCR	+99999.99
$**,**9.99	+99999.99

Sample

```
identification division.
program-id. example.

data division.
working-storage section.

01 alg-1 pic 9 value 0.
01 alg-5 pic 9(5) value 0.
01 alg-10 pic s9(10) value 0.
```

```
01 alg-1b pic 9 binary value 0.
01 alg-5b pic 9(5) binary value 0.
01 alg-10b pic s9(10) binary value 0.
01 alg-1e pic 9/ value 0.
01 alg-5e pic 999/99 value 0.
01 alg-10e pic -99/99/99/99/99 value 0.
01 alg-1d pic 9v99 binary value 0.
01 alg-5d pic 9(5)v99 binary value 0.
01 alg-10d pic s9(10)v99 binary value 0.

procedure division.
main-paragraph.
   display "FUNCTION HIGHEST-ALGEBRAIC(numeric):" upon sysout
   display FUNCTION HIGHEST-ALGEBRAIC(alg-1) upon sysout
   display FUNCTION HIGHEST-ALGEBRAIC(alg-5) upon sysout
   display FUNCTION HIGHEST-ALGEBRAIC(alg-10) upon sysout
   display FUNCTION HIGHEST-ALGEBRAIC(alg-1b) upon sysout
   display FUNCTION HIGHEST-ALGEBRAIC(alg-5b) upon sysout
   display FUNCTION HIGHEST-ALGEBRAIC(alg-10b) upon sysout
   display FUNCTION HIGHEST-ALGEBRAIC(alg-1e) upon sysout
   display FUNCTION HIGHEST-ALGEBRAIC(alg-5e) upon sysout
   display FUNCTION HIGHEST-ALGEBRAIC(alg-10e) upon sysout
   display FUNCTION HIGHEST-ALGEBRAIC(alg-1d) upon sysout
   display FUNCTION HIGHEST-ALGEBRAIC(alg-5d) upon sysout
   display FUNCTION HIGHEST-ALGEBRAIC(alg-10d) upon sysout
```

Sample Output

```
FUNCTION HIGHEST-ALGEBRAIC(numeric):
9
99999
9999999999
9
99999
9999999999
9
99999
9999999999
9.99
99999.99
9999999999.99
```

INTEGER Function

The INTEGER function returns the greatest integer value that is less than or equal to the argument.

The type of this function is integer.

General format

FUNCTION INTEGER (argument-1)

Arguments

Argument-1 shall be class numeric.

Returned values

The returned value is the greatest integer less than or equal to the value of argument-1. For example, if the value of argument-1 is -1.5, -2 is returned. If the value of argument-1 is +1.5, +1 is returned.

Sample

```
display "FUNCTION INTEGER(numeric):" upon sysout
display FUNCTION INTEGER(123.456) upon sysout
display FUNCTION INTEGER(123) upon sysout
display FUNCTION INTEGER(.123) upon sysout
display FUNCTION INTEGER(-.123) upon sysout
display FUNCTION INTEGER(-123) upon sysout
display FUNCTION INTEGER(-123.456) upon sysout
```

Sample Output

```
FUNCTION INTEGER(numeric):
123.0
123.0
0.0
-1.0
-123.0
-124.0
```

INTEGER-OF-DATE Function

The INTEGER-OF-DATE function converts a date in the Gregorian calendar from standard date form (YYYYMMDD) to integer date form.

The type of this function is integer.

General format

FUNCTION INTEGER-OF-DATE (argument-1)

Arguments

Argument-1 shall be an integer of the form YYYYMMDD, whose value is obtained from the calculation (YYYY * 10,000) + (MM * 100) + DD.

1. YYYY represents the year in the Gregorian calendar. It shall be an integer greater than 1600 and less than 10000.
2. MM represents a month and shall be a positive integer less than 13.
3. DD represents a day and shall be a positive integer less than 32 provided that it is valid for the specified month and year combination.

Returned values

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.

Sample

```
display "FUNCTION INTEGER-OF-DATE(YYYYMMDD):" upon sysout
display FUNCTION INTEGER-OF-DATE(17760704) upon sysout
display FUNCTION INTEGER-OF-DATE(19690720) upon sysout
display FUNCTION INTEGER-OF-DATE(19711211) upon sysout
```

```
display FUNCTION INTEGER-OF-DATE(19720806) upon sysout
display FUNCTION INTEGER-OF-DATE(20000101) upon sysout
display FUNCTION INTEGER-OF-DATE(20010101) upon sysout
display FUNCTION INTEGER-OF-DATE(21230704) upon sysout
```

Sample Output

```
FUNCTION INTEGER-OF-DATE(YYYYMMDD):
64103
134610
135484
135723
145732
146098
190841
```

INTEGER-OF-DAY Function

The INTEGER-OF-DAY function converts a date in the Gregorian calendar from Julian date form (YYYYDDD) to integer date form.

The type of this function is integer.

General format

FUNCTION INTEGER-OF-DAY (argument-1)

Arguments

Argument-1 shall be an integer of the form YYYYDDD, whose value is obtained from the calculation (YYYY * 1000) + DDD.

1. YYYY represents the year in the Gregorian calendar. It shall be an integer greater than 1600 and less than 10000.
2. DDD represents the day of the year. It shall be a positive integer less than 367 provided that it is valid for the year specified.

Returned values

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.

Sample

```
display "FUNCTION INTEGER-OF-DAY(YYYYMMDD):" upon sysout
display FUNCTION INTEGER-OF-DAY(1776050) upon sysout
display FUNCTION INTEGER-OF-DAY(1969100) upon sysout
display FUNCTION INTEGER-OF-DAY(1971150) upon sysout
display FUNCTION INTEGER-OF-DAY(1972200) upon sysout
display FUNCTION INTEGER-OF-DAY(2000250) upon sysout
display FUNCTION INTEGER-OF-DAY(2001300) upon sysout
display FUNCTION INTEGER-OF-DAY(2123350) upon sysout
```

Sample Output

```
FUNCTION INTEGER-OF-DAY(YYYYMMDD):
63967
134509
135289
```

135704
145981
146397
191006

INTEGER-PART Function

The INTEGER-PART function returns an integer that is the integer portion of argument-1.

The type of this function is integer.

General format

FUNCTION INTEGER-PART (argument-1)

Arguments

Argument-1 shall be class numeric.

Returned values

The equivalent arithmetic expression shall be:

(FUNCTION SIGN (argument-1) * FUNCTION INTEGER (FUNCTION ABS (argument-1)))

where the arguments for the SIGN and ABS functions are the same as for the INTEGER-PART function itself.

NOTE - If the value of argument-1 is +1.5, +1 is returned; if the value of argument-1 is -1.5, -1 is returned; and if the value of argument-1 is zero, zero is returned.

Sample

```
display "FUNCTION INTEGER-PART(numeric):" upon sysout
display FUNCTION INTEGER-PART(123.456) upon sysout
display FUNCTION INTEGER-PART(123) upon sysout
display FUNCTION INTEGER-PART(.123) upon sysout
display FUNCTION INTEGER-PART(-.123) upon sysout
display FUNCTION INTEGER-PART(-123) upon sysout
display FUNCTION INTEGER-PART(-123.456) upon sysout
```

Sample Output

```
FUNCTION INTEGER-PART(numeric):
123.0
123.0
0.0
-0.0
-123.0
-123.0
```

LENGTH Function

The LENGTH function returns an integer equal to the length of the argument in alphanumeric character positions, national character positions, or Boolean positions, depending on the class of the argument.

The type of this function is integer.

General format

FUNCTION LENGTH (argument-1)

Arguments

Argument-1 shall be alphanumeric or national literal; a data item of any class or category.

Returned values

1. If argument-1 is an elementary data item of class national, is a national literal, or is a type declaration of category national, the returned value shall be an integer equal to the length of argument-1 in national character positions.
2. Otherwise, if argument-1 is an elementary data item, is an alphanumeric literal, or is an elementary type declaration, the returned value shall be an integer equal to the length of argument-1 in alphanumeric character positions.
3. If argument-1 is a group data item:
   a. If argument-1 or any data item subordinate to argument-1 is described with the DEPENDING phrase of the OCCURS clause:
      - The returned value shall be an integer equal to the length of argument-1 in alphanumeric character positions, as a sending data item, determined by evaluation of the data item specified in the DEPENDING phrase in accordance with the rules of the OCCURS clause. The contents of the data item specified in the DEPENDING phrase are used at the time the LENGTH function is evaluated.

   b. Otherwise, the returned value shall be an integer equal to the length of argument-1 in alphanumeric character positions.
      - The returned length shall include the number of implicit FILLER positions, if any, in argument-1.
4. When argument-1 does not occupy an integral number of alphanumeric character positions, the returned value shall be rounded to the next larger integer value.

Sample

```
identification division.
program-id. example.

data division.
working-storage section.

01 national-id pic n(10).

01 alg-1 pic 9 value 0.
01 alg-5 pic 9(5) value 0.
01 alg-10 pic s9(10) value 0.
01 alg-1b pic 9 binary value 0.
01 alg-5b pic 9(5) binary value 0.
01 alg-10b pic s9(10) binary value 0.
```

```
01 alg-1e pic 9/ value 0.
01 alg-5e pic 999/99 value 0.
01 alg-10e pic -99/99/99/99/99 value 0.
01 alg-1d pic 9v99 binary value 0.
01 alg-5d pic 9(5)v99 binary value 0.
01 alg-10d pic s9(10)v99 binary value 0.

procedure division.
main-paragraph.
   display "FUNCTION LENGTH(argument):" upon sysout
   display FUNCTION LENGTH(national-id) upon sysout
   display FUNCTION LENGTH(alg-1) upon sysout
   display FUNCTION LENGTH(alg-5) upon sysout
   display FUNCTION LENGTH(alg-10) upon sysout
   display FUNCTION LENGTH(alg-1b) upon sysout
   display FUNCTION LENGTH(alg-5b) upon sysout
   display FUNCTION LENGTH(alg-10b) upon sysout
   display FUNCTION LENGTH(alg-1e) upon sysout
   display FUNCTION LENGTH(alg-5e) upon sysout
   display FUNCTION LENGTH(alg-10e) upon sysout
   display FUNCTION LENGTH(alg-1d) upon sysout
   display FUNCTION LENGTH(alg-5d) upon sysout
   display FUNCTION LENGTH(alg-10d) upon sysout
```

Sample Output

```
FUNCTION LENGTH(argument):
10
1
5
10
2
4
8
2
6
15
2
4
8
```

LENGTH-AN Function

The LENGTH-AN function returns an integer equal to the length of the argument in number of alphanumeric character positions.

The type of the function is integer.

General format

FUNCTION LENGTH-AN (argument-1)

Arguments

Argument-1 shall be an alphanumeric or national literal; a data item of any class or category.

Returned values

1. If argument-1 is an elementary data item, is a literal, or is an elementary type declaration, the returned value shall be an integer equal to the length of argument-1 in alphanumeric character positions.
2. If argument-1 is a group data item:
    a. If argument-1 or any data item subordinate to argument-1 is described with the DEPENDING phrase of the OCCURS clause:
        - The returned value shall be an integer equal to the length of argument-1 in alphanumeric character positions, as a sending data item, determined by evaluation of the data item specified in the DEPENDING phrase in accordance with the rules of the OCCURS clause. The contents of the data item specified in the DEPENDING phrase are used at the time the LENGTH function is evaluated.
        - Otherwise, the value returned shall be an integer equal to the length of argument-1 in alphanumeric character positions.
        - The returned length shall include the number of implicit FILLER positions, if any, in argument-1.
3. When argument-1 does not occupy an integral number of alphanumeric character positions, the returned value shall be rounded to the next larger integer value.

Sample

```
identification division.
program-id. example.

data division.
working-storage section.

01 national-id pic n(10).

01 alg-1 pic 9 value 0.
01 alg-5 pic 9(5) value 0.
01 alg-10 pic s9(10) value 0.
01 alg-1b pic 9 binary value 0.
01 alg-5b pic 9(5) binary value 0.
01 alg-10b pic s9(10) binary value 0.
01 alg-1e pic 9/ value 0.
01 alg-5e pic 999/99 value 0.
01 alg-10e pic -99/99/99/99/99 value 0.
01 alg-1d pic 9v99 binary value 0.
01 alg-5d pic 9(5)v99 binary value 0.
01 alg-10d pic s9(10)v99 binary value 0.

procedure division.
main-paragraph.
   display "FUNCTION LENGTH-AN(argument):" upon sysout
   display FUNCTION LENGTH-AN(national-id) upon sysout
   display FUNCTION LENGTH-AN(alg-1) upon sysout
   display FUNCTION LENGTH-AN(alg-5) upon sysout
   display FUNCTION LENGTH-AN(alg-10) upon sysout
```

```
display FUNCTION LENGTH-AN(alg-1b) upon sysout
display FUNCTION LENGTH-AN(alg-5b) upon sysout
display FUNCTION LENGTH-AN(alg-10b) upon sysout
display FUNCTION LENGTH-AN(alg-1e) upon sysout
display FUNCTION LENGTH-AN(alg-5e) upon sysout
display FUNCTION LENGTH-AN(alg-10e) upon sysout
display FUNCTION LENGTH-AN(alg-1d) upon sysout
display FUNCTION LENGTH-AN(alg-5d) upon sysout
display FUNCTION LENGTH-AN(alg-10d) upon sysout
```

Sample Output

```
FUNCTION LENGTH-AN(argument):
20
1
5
10
2
4
8
2
6
15
2
4
8
```

LINE Function

The LINE function returns the line number currently being compiled to the nearest accuracy; it will actually return the line number containing the following token. It may be used for embedding messages about the source file itself within the executable. This function is resolved at compile-time, not run-time.

General format

FUNCTION LINE

Returned values

The returned value is the filename of the currently compiling file.

Sample Program

```
identification division.
program-id. sample_funcs.

procedure division.
main-paragraph.

    display "Compiling file '" function file "'" upon sysout

    display "Compiling line " function line "." upon sysout
    display "Compiling line " function line "." upon sysout
    display "Compiling line " function line "." upon sysout
    display "Compiling line " function line "." upon sysout
```

```
display "Compiling line " function line "." upon sysout

display "Compiling column " function column "." upon sysout
display "Compiling column " function column "." upon sysout
display "Compiling column " function column "." upon sysout
display "Compiling column " function column "." upon sysout
display "Compiling column " function column "." upon sysout
.
```

Sample Output

```
Compiling file 'sample_funcs.cbl'
Compiling line 9.
Compiling line 10.
Compiling line 11.
Compiling line 12.
Compiling line 13.
Compiling column 49.
Compiling column 49.
Compiling column 49.
Compiling column 49.
Compiling column 49.
```

LOCALE-DATE Function

The LOCALE-DATE functions returns a text string in locale-dependent form.

General format

FUNCTION LOCALE-DATE (argument-1 [LOCALE locale-name])

Arguments

Argument-1 must be in the same format as returned by the CURRENT-DATE function, positions 1 through 8.

Returned values

The locale-dependent text string is returned.

Sample

```
id division.
program-id. localedate.

 data division.
 working-storage section.

    01 date-group.
*                       "@Y-%m-%d"
       05 plain-date format of date.

*                       "%H:%M:%S"
   01 time-group.
       05 plain-time format of time.

*                       "@Y-%m-%d-%H.%M.%S.@Sm"
```

```
 01 timestamp-group.
    05 plain-timestamp format of timestamp.
     05 other-timestamp format of timestamp.

procedure division.
main-paragraph.

    move "1972-08-06" to plain-date
    move "12:34:56" to plain-time
    move "1972-08-06-12.34.56.123456" to plain-timestamp
     move "1972-09-08-16.11.22.333333" to other-timestamp
     .

LOCALE-DATE.

   display "" upon sysout
     display "LOCALE-DATE:" upon sysout
     display "" upon sysout

     display function locale-date("19720806")
        upon sysout
     display function locale-date(
        "19720806" locale
        )
        upon sysout
     display function locale-date(
        "19720806" locale "en_US"
        )
        upon sysout
     display function locale-date(
        "19720806" locale "en_US:S"
        )
        upon sysout
     display function locale-date(
        "19720806" locale "en_US:M"
        )
        upon sysout
     display function locale-date(
        "19720806" locale "en_US:L"
        )
        upon sysout
     display function locale-date(
        "19720806" locale "en_US:F"
        )
        upon sysout

     display function locale-date(
        "19720806" locale "de_DE"
        )
        upon sysout
     display function locale-date(
        "19720806" locale "de_DE:S"
        )
        upon sysout
     display function locale-date(
        "19720806" locale "de_DE:M"
```

```
    )
    upon sysout
  display function locale-date(
    "19720806" locale "de_DE:L"
    )
    upon sysout
  display function locale-date(
    "19720806" locale "de_DE:F"
    )
    upon sysout
  .
```

Sample Output

```
LOCALE-DATE:
8/6/72
8/6/72
8/6/72
8/6/72
Aug 6, 1972
August 6, 1972
Sunday, August 6, 1972
06.08.72
06.08.72
06.08.1972
6. August 1972
Sonntag, 6. August 1972
```

LOCALE-TIME Function

The LOCALE-TIME function returns a text string in locale-dependent form.

General format

FUNCTION LOCALE-TIME (argument-1 [LOCALE locale-name])

Arguments

Argument-1 must be in the same format as returned by the CURRENT-DATE function, positions 9 through 21.

Returned values

The locale-dependent text string is returned.

Sample

```
id division.
     program-id. localetime.

     data division.
     working-storage section.

     01 date-group.
    *                    "@Y-%m-%d"
        05 plain-date format of date.

    *                    "%H:%M:%S"
```

```
       01 time-group.
          05 plain-time format of time.

      *                        "@Y-%m-%d-%H.%M.%S.@Sm"
       01 timestamp-group.
          05 plain-timestamp format of timestamp.
        05 other-timestamp format of timestamp.

       procedure division.
       main-paragraph.

          move "1972-08-06" to plain-date
          move "12:34:56" to plain-time
          move "1972-08-06-12.34.56.123456" to plain-timestamp
        move "1972-09-08-16.11.22.333333" to other-timestamp
        .
       LOCALE-TIME.
          display "" upon sysout
        display "LOCALE-TIME:" upon sysout
        display "" upon sysout

        display function locale-time(
           "184812"
    )
           upon sysout
        display function locale-time(
           "184812" locale
           )
           upon sysout
        display function locale-time(
           "184812" locale "en_US"
           )
           upon sysout
        display function locale-time(
           "184812" locale "en_US:S"
           )
           upon sysout
        display function locale-time(
           "184812" locale "en_US:M"
           )
           upon sysout
        display function locale-time(
           "184812" locale "en_US:L"
           )
           upon sysout
        display function locale-time(
           "184812" locale "en_US:F"
           )
           upon sysout
    .
```

Sample Output

```
LOCALE-TIME:

6:48 PM
6:48 PM
```

6:48 PM
6:48 PM
6:48:12 PM
6:48:12 PM PDT
6:48:12 PM PDT

LOG Function

The LOG function returns a numeric value that approximates the logarithm to the base e (natural log) of argument-1.

The type of this function is numeric.

General format

FUNCTION LOG (argument-1)

Arguments

1. Argument-1 shall be class numeric.
2. The value of argument-1 shall be greater than zero.

Returned values

The returned value is the approximation of the logarithm to the base e of argument-1.

Sample

```
display "FUNCTION LOG(numeric):" upon sysout
display FUNCTION LOG(1000) upon sysout
display FUNCTION LOG(100) upon sysout
display FUNCTION LOG(1) upon sysout
display FUNCTION LOG(.123) upon sysout
```

Sample Output

```
FUNCTION LOG(numeric):
6.907755278982137
4.605170185988092
0.0
-2.0955709236097197
```

LOG10 Function

The LOG10 function returns a numeric value that approximates the logarithm to the base 10 of argument-1.

The type of this function is numeric.

General format

FUNCTION LOG10 (argument-1)

Arguments

1. Argument-1 shall be class numeric.

2. The value of argument-1 shall be greater than zero.

Returned values

The returned value is the approximation of the logarithm to the base 10 of argument-1.

Sample

```
display "FUNCTION LOG10(numeric):" upon sysout
display FUNCTION LOG10(1000) upon sysout
display FUNCTION LOG10(100) upon sysout
display FUNCTION LOG10(1) upon sysout
display FUNCTION LOG10(.123) upon sysout
```

Sample Output

```
FUNCTION LOG10(numeric):
2.9999999999999996
2.0
0.0
-0.9100948885606021
```

LOWER-CASE Function

The LOWER-CASE function returns a character string that is the same length as argument-1 with each uppercase letter replaced by the corresponding lowercase letter.

The type of the function depends on the argument type as follows:

Argument Type	Function Type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
National	National

General format

FUNCTION LOWER-CASE (argument-1)

Arguments

Argument-1 shall be class alphabetic, alphanumeric, or national and shall be at least one character position in length.

Returned values

1. The same character string as argument-1 is returned, except that each uppercase letter shall be replaced by the corresponding lowercase letter.
2. The character string returned has the same length as argument-1.
3. If the computer's character set does not include lowercase letters, no changes take place in the character string. Elastic COBOL always performs lower-case conversion.

Sample

```
display "FUNCTION LOWER-CASE(alphanumeric):" upon sysout
display FUNCTION LOWER-CASE("UPPER CASE") upon sysout
display FUNCTION LOWER-CASE("lower case") upon sysout
display FUNCTION LOWER-CASE("MiXeD CaSe") upon sysout
```

Sample Output

FUNCTION LOWER-CASE(alphanumeric):
upper case
lower case
mixed case

LOWEST-ALGEBRAIC Function

The LOWEST-ALGEBRAIC function returns a value that is equal to the lowest algebraic value that may be represented in argument-1.

The type of this function depends upon the argument types as follows:

Argument Type	Function Type
Integer	Integer
Numeric	Numeric
Numeric-edited	Numeric

General format

FUNCTION LOWEST-ALGEBRAIC (argument-1)

Arguments

Arguments-1 shall be an elementary data item of category numeric or numeric-edited.

Returned values

The value returned is equal to the lowest algebraic value that may be represented in argument-1.

NOTE - The following illustrates the expected results for some values of argument-1.

Argument-1 characteristics	Value returned
S999	-999
S9(4) BINARY	-9999
99V9(3)	0
$**,**9.99BCR	-99999.99
$**,**9.99	0

Sample

```
identification division.
program-id. example.

data division.
working-storage section.

01 alg-1 pic 9 value 0.
01 alg-5 pic 9(5) value 0.
01 alg-10 pic s9(10) value 0.
01 alg-1b pic 9 binary value 0.
01 alg-5b pic 9(5) binary value 0.
01 alg-10b pic s9(10) binary value 0.
01 alg-1e pic 9/ value 0.
01 alg-5e pic 999/99 value 0.
01 alg-10e pic -99/99/99/99/99 value 0.
01 alg-1d pic 9v99 binary value 0.
```

```
01 alg-5d pic 9(5)v99 binary value 0.
01 alg-10d pic s9(10)v99 binary value 0.

procedure division.
main-paragraph.
   display "FUNCTION LOWEST-ALGEBRAIC(numeric):" upon sysout
   display FUNCTION LOWEST-ALGEBRAIC(alg-1) upon sysout
   display FUNCTION LOWEST-ALGEBRAIC(alg-5) upon sysout
   display FUNCTION LOWEST-ALGEBRAIC(alg-10) upon sysout
   display FUNCTION LOWEST-ALGEBRAIC(alg-1b) upon sysout
   display FUNCTION LOWEST-ALGEBRAIC(alg-5b) upon sysout
   display FUNCTION LOWEST-ALGEBRAIC(alg-10b) upon sysout
   display FUNCTION LOWEST-ALGEBRAIC(alg-1e) upon sysout
   display FUNCTION LOWEST-ALGEBRAIC(alg-5e) upon sysout
   display FUNCTION LOWEST-ALGEBRAIC(alg-10e) upon sysout
   display FUNCTION LOWEST-ALGEBRAIC(alg-1d) upon sysout
   display FUNCTION LOWEST-ALGEBRAIC(alg-5d) upon sysout
   display FUNCTION LOWEST-ALGEBRAIC(alg-10d) upon sysout
```

Sample Output

```
FUNCTION LOWEST-ALGEBRAIC(numeric):
0
0
-9999999999
0
0
-9999999999
0
0
-9999999999
0
0
-9999999999.99
```

MAX Function

The MAX function returns the content of the argument-1 that contains the maximum value.

The type of this function depends upon the argument types as follows:

Argument Type	Function Type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
National	National
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

General format

FUNCTION MAX ({argument-1} …)

Arguments

1. Argument-1 shall be of any class except object or pointer.

2. If more than one argument-1 is specified, all arguments shall be of the same class with the exception that mixing of arguments of alphabetic and alphanumeric classes is allowed.

Returned values

1. The returned value is the content of the argument-1 having the greatest value. The comparisons used to determine the greatest value are made according to the rules for simple conditions. (See Simple conditions.)
2. If more than one argument-1 has the same greatest value, the content of the argument-1 returned is the leftmost argument-1 having that value.
3. If the type of the function is alphanumeric or national, the size of the returned value is the same as the size of the selected argument-1.

Sample

```
display "FUNCTION MAX(...):" upon sysout
display FUNCTION MAX( 1 2 3 4 ) upon sysout
display FUNCTION MAX( 4 3 2 1 ) upon sysout
display FUNCTION MAX( 123.456 123.231 123.855 123.323 ) upon sysout
display FUNCTION MAX( "A" "B" "C" "D" ) upon sysout
display FUNCTION MAX( "D" "C" "B" "A" ) upon sysout
display FUNCTION MAX( "Brian" "Bryan" "Brent" "Brianne" ) upon sysout
```

Sample Output

```
FUNCTION MAX(...):
4
4
123.855
D
D
Bryan
```

MEAN Function

The MEAN function returns a numeric value that is the arithmetic mean (average) of its arguments.

The type of this function is numeric.

General format

FUNCTION MEAN ({argument-1} ...)

Arguments

Argument-1 shall be class numeric.

Returned values

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-11 + argument-12) / 2)

3. For n occurrences of argument-1,

   ((argument-11 + argument-12 +... + argument-1n) / n)

Sample

```
display "FUNCTION MEAN(...):" upon sysout
display FUNCTION MEAN( 1 2 3 4 5 6 7 8 9 10 ) upon sysout
display FUNCTION MEAN( 139 34.5 345.5 864.435 384 394 945 ) upon sysout
display FUNCTION MEAN( 93 83 71 23 2 83 98 34 1 7 ) upon sysout
```

Sample Output

```
FUNCTION MEAN(...):
5.5
443.776428571428571428
49.5
```

MEDIAN Function

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 type of this function is numeric.

General format

FUNCTION MEDIAN ({argument-1} …)

Arguments

Argument-1 shall be class numeric.

Returned values

1. When the number of occurrences of argument-1 is odd, the returned value shall be 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. For the purposes of the equivalent arithmetic expression, the middle value is referred to as argument-a. The equivalent arithmetic expression shall be (argument-a)
2. When the number of occurrences of argument-1 is even, the returned value is the arithmetic mean of the two middle values. For the purposes of the equivalent arithmetic expression, the two middle values are referred to as argument-b and argument-c. The equivalent arithmetic expression shall be ((argument-b + argument-c) / 2)
3. The comparisons used to arrange the argument-1 values in sorted order are made according to the rules for simple conditions. (See Simple conditions.)

Sample

```
display "FUNCTION MEDIAN(...):" upon sysout
display FUNCTION MEDIAN( 1 2 3 4 5 6 7 8 9 10 ) upon sysout
display FUNCTION MEDIAN( 139 34.5 345.5 864.435 384 394 945 ) upon sysout
display FUNCTION MEDIAN( 93 83 71 23 2 83 98 34 1 7 ) upon sysout
```

Sample Output

```
FUNCTION MEDIAN(...):
```

5.5
384
52.5

MIDRANGE Function

The MIDRANGE (middle range) function returns a numeric value that is the arithmetic mean (average) of the values of the minimum argument and the maximum argument.

The type of this function is numeric.

General format

FUNCTION MIDRANGE ({argument-1} …)

Arguments

Argument-1 shall be class numeric.

Returned values

The equivalent arithmetic expression shall be

((FUNCTION MAX (argument-1) + FUNCTION MIN (argument-1)) / 2)

where the arguments for the MAX and MIN functions are the same as the arguments for the MIDRANGE function itself.

Sample

```
display "FUNCTION MIDRANGE(...):" upon sysout
display FUNCTION MIDRANGE( 1 2 3 4 5 6 7 8 9 10 ) upon sysout
display FUNCTION MIDRANGE( 139 34.5 345.5 864.435 384 394 945 ) upon sysout
display FUNCTION MIDRANGE( 93 83 71 23 2 83 98 34 1 7 ) upon sysout
```

Sample Output

```
FUNCTION MIDRANGE(...):
5.5
489.75
49.5
```

MIN Function

The MIN function returns the content of the argument-1 that contains the minimum value.

The type of this function depends upon the argument types as follows:

Argument Type	Function Type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
National	National
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

General format

FUNCTION MIN ({argument-1} …)

Arguments

1. Argument-1 shall be of any class except object or pointer.
2. If more than one argument-1 is specified, all arguments shall be of the same class with the exception that mixing of arguments of alphabetic and alphanumeric classes is allowed.

Returned values

1. The returned value is the content of the argument-1 having the least value. The comparisons used to determine the least value are made according to the rules for simple conditions. (See Simple conditions.)
2. If more than one argument-1 has the same least value, the content of the argument-1 returned is the leftmost argument-1 having that value.
3. If the type of the function is alphanumeric or national, the size of the returned value is the same as the size of the selected argument-1.

Sample

```
display "FUNCTION MIN(...):" upon sysout
display FUNCTION MIN( 1 2 3 4 ) upon sysout
display FUNCTION MIN( 4 3 2 1 ) upon sysout
display FUNCTION MIN( 123.456 123.231 123.855 123.323 ) upon sysout
display FUNCTION MIN( "A" "B" "C" "D" ) upon sysout
display FUNCTION MIN( "D" "C" "B" "A" ) upon sysout
display FUNCTION MIN( "Brian" "Bryan" "Brent" "Brianne" ) upon sysout
```

Sample Output

```
FUNCTION MIN(...):
1
1
123.231
A
A
Brent
```

MOD Function

The MOD function returns an integer value that is argument-1 modulo argument-2.

The type of this function is integer.

General format

FUNCTION MOD (argument-1 argument-2)

Arguments

1. Argument-1 and argument-2 shall be integers.
2. The value of argument-2 shall not be zero.

Returned values

The equivalent arithmetic expression shall be

(argument-1 - (argument-2 * FUNCTION INTEGER (argument-1 / argument-2)))

where argument-1 and argument-2 for the INTEGER function are the same as the arguments for the MOD function itself.

NOTE - 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

Sample

```
display "FUNCTION MOD(intger integer):" upon sysout
display FUNCTION MOD(100 25) upon sysout
display FUNCTION MOD(100 24) upon sysout
display FUNCTION MOD(100 23) upon sysout
display FUNCTION MOD(100 22) upon sysout
display FUNCTION MOD(100 21) upon sysout
display FUNCTION MOD(100 20) upon sysout
display FUNCTION MOD(100 3) upon sysout
display FUNCTION MOD(100 2) upon sysout
display FUNCTION MOD(100 1) upon sysout
display FUNCTION MOD(-100 25) upon sysout
display FUNCTION MOD(100 -24) upon sysout
display FUNCTION MOD(-100 23) upon sysout
display FUNCTION MOD(100 -22) upon sysout
display FUNCTION MOD(-100 21) upon sysout
display FUNCTION MOD(100 -20) upon sysout
display FUNCTION MOD(-100 3) upon sysout
display FUNCTION MOD(100 -2) upon sysout
display FUNCTION MOD(-100 1) upon sysout
```

Sample Output

```
FUNCTION MOD(intger integer):
0
4
8
12
16
0
1
0
0
0
-20
15
-10
5
0
2
0
0
```

NATIONAL-OF Function

The NATIONAL-OF function returns a character string containing the national character internal representation of the characters in the argument.

The type of the function is national.

General format

FUNCTION NATIONAL-OF (argument-1 [argument-2])

Arguments

1. Argument-1 shall be of class alphabetic or class alphanumeric. If alphanumeric, it may contain alphanumeric characters or national characters in external media format, or both.
2. Argument-2 shall be of category national and shall be one character in length. Argument-2 specifies a national substitution character for use in conversion of alphanumeric characters for which there is no corresponding national character.

Returned values

1. A character string shall be returned with each alphanumeric character and each national character in argument-1 converted to its corresponding national internal representation. Any control functions used for external media format in argument-1 shall be recognized only for purposes of distinguishing characters, and shall not be considered as separate characters of argument-1.
2. If argument-2 is specified, each character in argument-1 that has no corresponding internal national representation shall be converted to the substitution character specified by argument-2.
3. If argument-2 is unspecified, each character that has no corresponding internal national representation shall be converted to a national space.
4. The length of the returned value shall be the number of character positions of usage national required to hold the converted argument and depends on the number of characters contained in argument-1.

Sample

```
display "FUNCTION NATIONAL-OF(alphanumeric):" upon sysout
display FUNCTION NATIONAL-OF("A") upon sysout
display FUNCTION NATIONAL-OF("B") upon sysout
display FUNCTION NATIONAL-OF("C") upon sysout
```

Sample Output

```
FUNCTION NATIONAL-OF(alphanumeric):
?
?
?
```

NUMVAL Function

The NUMVAL function returns the numeric value represented by the character string specified by argument-1. Leading and trailing spaces are ignored.

The type of this function is numeric.

General format

FUNCTION NUMVAL (argument-1)

Arguments

1. Argument-1 shall be an alphanumeric or national literal or an alphanumeric or national data item whose content has one of the following two formats:

   [space] [+|-] [space] { {digit [. [digit]]} | {. digit}} [space]

   or

   [space] { {digit [. [digit]] | { . digit}} [space] [+|-|CR|DB] [space]

   where space is a string of zero or more spaces and digit is a string of one to 31 digits.

2. The total number of digits in argument-1 shall not exceed 31.
3. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, a comma shall be used in argument-1 rather than a decimal point.

Returned values

1. The returned value is the numeric value represented by argument-1.
2. The number of digits returned is at most 18.

Sample

```
display "FUNCTION NUMVAL(alphanumeric):" upon sysout
display FUNCTION NUMVAL("  123.456  ") upon sysout
display FUNCTION NUMVAL("  123  ") upon sysout
display FUNCTION NUMVAL("  .456  ") upon sysout
```

Sample Output

```
FUNCTION NUMVAL(alphanumeric):
123.456
123
.456
```

NUMVAL-C Function

The NUMVAL-C function returns the numeric value represented by the character string specified by argument-1. Any optional currency sign specified by argument-2 and any optional commas preceding the decimal point are ignored.

The type of this function is numeric.

General format

FUNCTION NUMVAL-C (argument-1 [argument-2])

Arguments

1. Argument-1 shall be an alphanumeric or national literal or an alphanumeric or national data item whose content has one of the following two formats:

[space] [+|-] [space] [cs] [space] { {digit [, digit] … [. [digit]]} | {. Digit}} [space]
or
[space] [cs] [space] { {digit [, digit] … [. [digit]] } | {. digit} } [space] [+|-|CR|DB] [space]

where space is a string of zero or more spaces, cs is the string of one or more characters specified by argument-2 and digit is a string of one or more digits.

2. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, the functions of the comma and decimal point in argument-1 are reversed.
3. The total number of digits in argument-1 shall not exceed 18.
4. Argument-2, if specified, shall be of the same class as argument-1.
5. If argument-2 is not specified, the character used for cs shall be the currency symbol specified for the program when argument-1 is of class alphanumeric; or, if argument-1 is of class national, the national representation of the currency symbol specified for the program. National representation of cs is the value returned from FUNCTION NATIONAL-OF(cs).

Returned values

1. The returned value is the numeric value represented by argument-1.
2. The number of digits returned is at most 18.

Sample

```
display "FUNCTION NUMVAL-C(alphanumeric):" upon sysout
display FUNCTION NUMVAL-C("  $123.456  ") upon sysout
display FUNCTION NUMVAL-C("  $123  ") upon sysout
display FUNCTION NUMVAL-C("  $.456  ") upon sysout
```

Sample Output

```
FUNCTION NUMVAL-C(alphanumeric):
123.456
123
.456
```

NUMVAL-F Function

The NUMVAL-F function returns the floating-point value represented by the character string specified by argument-1. Leading, trailing, and embedded spaces are ignored.

The type of this function is numeric.

General format

FUNCTION NUMVAL-F (argument-1)

Arguments

1. Argument-1 shall be an alphanumeric or national literal or an alphanumeric or national data item whose content has the following format:

[space] [+|-] [space] {{ digit [. [digit]] } | {. digit}} [space] [E [space] {+|-} [space] n [space]]

where space is a string of zero or more spaces; n is one, two, or three digits representing the exponent; and digit is a string of one to 18 digits.

2. The total number of digits in the significant shall not exceed 18.
3. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, a comma shall be used in argument-1 rather than a decimal point.

Returned values

1. Leading, trailing, and embedded spaces are ignored.
2. A floating-point data item is returned that contains the numeric value represented by argument-1.

Sample

```
display "FUNCTION NUMVAL-F(alphanumeric):" upon sysout
display FUNCTION NUMVAL-F("  123.456  ") upon sysout
display FUNCTION NUMVAL-F("  123  ") upon sysout
display FUNCTION NUMVAL-F("  .456  ") upon sysout
```

Sample Output

```
FUNCTION NUMVAL-F(alphanumeric):
123.456
123.0
0.456
```

ORD Function

The ORD function returns an integer value that is the ordinal position of argument-1 in the collating sequence for the program. The lowest ordinal position is 1.

The type of this function is integer.

General format

FUNCTION ORD (argument-1)

Arguments

Argument-1 shall be of one character position in length and shall be of class alphabetic, alphanumeric, or national.

Returned values

1. If the class of argument-1 is alphabetic or alphanumeric, the returned value shall be the ordinal position of argument-1 in the current alphanumeric program collating sequence.
2. If the class of argument-1 is national, the returned value shall be the ordinal position of argument-1 in the current national program collating sequence.

Sample

```
display "FUNCTION ORD(alphanumeric):" upon sysout
display FUNCTION ORD("A") upon sysout
display FUNCTION ORD("B") upon sysout
display FUNCTION ORD("c") upon sysout
display FUNCTION ORD("d") upon sysout
```

Sample Output

```
FUNCTION ORD(alphanumeric):
66
67
100
```

ORD-MAX Function

The ORD-MAX function returns a value that is the ordinal number of the argument-1 that contains the maximum value.

The type of this function is integer.

General format

FUNCTION ORD-MAX ({argument-1} …)

Arguments

1. Argument-1 shall be of any class except object or pointer.
2. If more than one argument-1 is specified, all arguments shall be of the same class with the exception that mixing of arguments of alphabetic and alphanumeric classes is allowed.

Returned values

1. The returned value is the ordinal number that corresponds to the position of the argument-1 having the greatest value in the argument-1 series.
2. The comparisons used to determine the greatest valued argument are made according to the rules for simple conditions. (See Simple conditions.)
3. If more than one argument-1 has the same greatest value, the number returned corresponds to the position of the leftmost argument-1 having that value.

Sample

```
display "FUNCTION ORD-MAX(...):" upon sysout
display FUNCTION ORD-MAX( 1 2 3 4 ) upon sysout
display FUNCTION ORD-MAX( 4 3 2 1 ) upon sysout
display FUNCTION ORD-MAX( 123.456 123.231 123.855 123.323 ) upon sysout
display FUNCTION ORD-MAX( "A" "B" "C" "D" ) upon sysout
display FUNCTION ORD-MAX( "D" "C" "B" "A" ) upon sysout
display FUNCTION ORD-MAX( "Brian" "Bryan" "Brent" "Brianne" ) upon sysout
```

Sample Output

```
FUNCTION ORD-MAX(...):
4
1
3
4
1
2
```

ORD-MIN Function

The ORD-MIN function returns a value that is the ordinal number of the argument that contains the minimum value.

The type of this function is integer.

General format

FUNCTION ORD-MIN ({argument-1} …)

Arguments

1. Argument-1 shall be of any class object or pointer.
2. If more than one argument-1 is specified, all arguments shall be of the same class with the exception that mixing of arguments of alphabetic and alphanumeric classes is allowed.

Returned values

1. The returned value is the ordinal number that corresponds to the position of the argument-1 having the least value in the argument-1 series.
2. The comparisons used to determine the least valued argument-1 are made according to the rules for simple conditions. (See Simple conditions.)
3. If more than one argument-1 has the same least value, the number returned corresponds to the position of the leftmost argument-1 having that value.

Sample

```
display "FUNCTION ORD-MIN(...):" upon sysout
display FUNCTION ORD-MIN( 1 2 3 4 ) upon sysout
display FUNCTION ORD-MIN( 4 3 2 1 ) upon sysout
display FUNCTION ORD-MIN( 123.456 123.231 123.855 123.323 ) upon sysout
display FUNCTION ORD-MIN( "A" "B" "C" "D" ) upon sysout
display FUNCTION ORD-MIN( "D" "C" "B" "A" ) upon sysout
display FUNCTION ORD-MIN( "Brian" "Bryan" "Brent" "Brianne" ) upon sysout
```

Sample Output

```
FUNCTION ORD-MIN(...):
1
4
2
1
4
3
```

PARAMETER Function

The PARAMETER function returns a command-line parameter by number. Each command-line argument is numbered from 0 to argument-length-1.

General format

FUNCTION PARAMETER (argument-1)

Arguments

Argument-1 shall be class numeric.

Returned values

The returned value is the value of the command line parameter in the designated parameter slot.

Sample Program

```
identification division.
program-id. sample_parameter.

procedure division.
main-paragraph.

    display "" upon sysout
    display "FUNCTION PARAMETER ( numeric ):" upon sysout
    display function parameter ( 0 ) upon sysout
    display function parameter ( 1 ) upon sysout
    display function argument-length upon sysout
    .
```

Sample Run

```
java sample_parameter alpha beta gamma delta
```

Sample Output

```
FUNCTION PARAMETER ( numeric ):
alpha
beta
4
```

PI Function

The PI function returns a value that is an approximation of π, the ratio of the circumference of a circle to its diameter.

The type of the function is numeric.

General format

FUNCTION PI

Returned values

The equivalent arithmetic expression shall be

(3 + .14159265358979323846264338327 95)

Sample

```
display "FUNCTION PI:" upon sysout
display FUNCTION PI upon sysout
```

Sample Output

```
FUNCTION PI:
3.141592653589793
```

PRESENT-VALUE Function

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 type of this function is numeric.

General format

FUNCTION PRESENT-VALUE (argument-1 {argument-2} …)

Arguments

1. Argument-1 and argument-2 shall be of the class numeric.
2. The value of argument-1 shall be greater than -1.

Returned values

The equivalent arithmetic expression shall be as follows:

1. For one occurrence of argument-2,
   (argument-2 / (1 + argument-1))
2. For two occurrences of argument-2,
   (argument-21 / (1 + argument-1) + argument-22 / (1 + argument-1) ** 2)
3. For n occurrences of argument-2, the equivalent arithmetic expression shall be
   (FUNCTION SUM (
   (argument-21 / (1 + argument-1) ** 1)
   ...
   (argument-2n / (1 + argument-1) ** n)))

where argument-1 and argument-2i in the terms of the SUM function are the same as the arguments for the PRESENT-VALUE function itself.

Sample

```
display "FUNCTION PRESENT-VALUE(numeric numeric...):" upon sysout
display FUNCTION PRESENT-VALUE(.1 1000 1100 1200) upon sysout
display FUNCTION PRESENT-VALUE(.1 1000 1100) upon sysout
display FUNCTION PRESENT-VALUE(.1 1000) upon sysout
display FUNCTION PRESENT-VALUE(.2 1000 1100 1200) upon sysout
display FUNCTION PRESENT-VALUE(.2 1000 1100) upon sysout
display FUNCTION PRESENT-VALUE(.2 1000) upon sysout
```

Sample Output

```
FUNCTION PRESENT-VALUE(numeric numeric...):
2719.759579263711
1818.181818181818
909.090909090909
2291.666666666667
1597.2222222222222
833.3333333333334
```

RANDOM Function

The RANDOM function returns a numeric value that is a pseudo-random number.

The type of this function is numeric.

General format

FUNCTION RANDOM [(argument-1)]

Arguments

1. If argument-1 is specified, it shall be zero or a positive integer. It is used as the seed value to generate a sequence of pseudo-random numbers.
2. If a subsequent reference specifies argument-1, a new sequence of pseudo-random numbers is started.
3. If the first reference to this function in the run unit does not specify argument-1, the seed value is currently .12923; this value is subject to change.
4. In each case, subsequent references without specifying argument-1 return the next number in the current sequence.

Returned values

1. The returned value is greater than or equal to zero and less than one.
2. For a given seed value on a given implementation, the sequence of pseudo-random numbers will always be the same.
3. The seed value may be any value distinct when represented as double-floating point.

Sample

```
display "FUNCTION RANDOM([numeric]):" upon sysout
display FUNCTION RANDOM upon sysout
display FUNCTION RANDOM upon sysout
display FUNCTION RANDOM upon sysout
display FUNCTION RANDOM upon sysout
display FUNCTION RANDOM(123) upon sysout
display FUNCTION RANDOM upon sysout
display FUNCTION RANDOM upon sysout
display FUNCTION RANDOM upon sysout
display FUNCTION RANDOM upon sysout
```

Sample Output

```
FUNCTION RANDOM([numeric]):
0.7071427150096409
0.5897318300267942
0.8797072458699503
0.4097427119110443
0.53116953637257
0.5330600801279523
0.04244432397374853
0.450866761969957
0.43319451861208025
```

RANGE Function

The RANGE function returns a value that is equal to the value of the maximum argument minus the value of the minimum argument.

The type of this function depends upon the argument types as follows:

Argument type	Function type
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

General format

FUNCTION RANGE ({argument-1} …)

Arguments

Argument-1 shall be class numeric.

Returned values

The equivalent arithmetic expression shall be

(FUNCTION MAX (argument-1) - FUNCTION MIN (argument-1))

where the arguments for the MAX and MIN functions are the same as the arguments for the RANGE function itself.

Sample

```
display "FUNCTION RANGE(...):" upon sysout
display FUNCTION RANGE( 1 2 3 4 ) upon sysout
display FUNCTION RANGE( 4 3 2 1 ) upon sysout
display FUNCTION RANGE( 123.456 123.231 123.855 123.323 ) upon sysout
```

Sample Output

```
FUNCTION RANGE(...):
3
3
0.624
```

REM Function

The REM function returns a numeric value that is the remainder of argument-1 divided by argument-2.

The type of this function is numeric.

General format

FUNCTION REM (argument-1 argument-2)

Arguments

1. Argument-1 and argument-2 shall be class numeric.
2. The value of argument-2 shall not be zero.

Returned values

The equivalent arithmetic expression shall be

(argument-1 - (argument-2 * FUNCTION INTEGER-PART (argument-1 / argument-2)))

where argument-1 and argument-2 of the INTEGER-PART function are the same as the arguments for the REM function itself.

Sample

```
display "FUNCTION REM(intger integer):" upon sysout
display FUNCTION REM(100 25) upon sysout
display FUNCTION REM(100 24) upon sysout
display FUNCTION REM(100 23) upon sysout
display FUNCTION REM(100 22) upon sysout
display FUNCTION REM(100 21) upon sysout
display FUNCTION REM(100 20) upon sysout
display FUNCTION REM(100 3) upon sysout
display FUNCTION REM(100 2) upon sysout
display FUNCTION REM(100 1) upon sysout
display FUNCTION REM(-100 25) upon sysout
display FUNCTION REM(100 -24) upon sysout
display FUNCTION REM(-100 23) upon sysout
display FUNCTION REM(100 -22) upon sysout
display FUNCTION REM(-100 21) upon sysout
display FUNCTION REM(100 -20) upon sysout
display FUNCTION REM(-100 3) upon sysout
display FUNCTION REM(100 -2) upon sysout
display FUNCTION REM(-100 1) upon sysout
```

Sample Output

```
FUNCTION REM(intger integer):
0
4
8
12
16
0
1
0
0
0
4
-8
12
-16
0
-1
0
0
```

REVERSE Function

The REVERSE function returns a character string of exactly the same length as argument-1 and whose characters are exactly the same as those of argument-1, except that they are in reverse order.

The type of the function depends on the argument type as follows:

Argument type	Function type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
National	National

General format

FUNCTION REVERSE (argument-1)

Arguments

Argument-1 shall be of class alphabetic, alphanumeric, or national and shall be at least one character position in length.

Returned values

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.

Sample

```
display "FUNCTION REVERSE(alphanumeric):" upon sysout
display FUNCTION REVERSE("Hello there!") upon sysout
display FUNCTION REVERSE("Heirloom") upon sysout
display FUNCTION REVERSE("FUNCTION REVERSE") upon sysout
```

Sample Output

```
FUNCTION REVERSE(alphanumeric):
!ereht olleH
moolrieH
ESREVER NOITCNUF
```

SIGN Function

The SIGN function returns +1, 0, or -1 depending on the sign of the argument.

The type of the function is integer.

General Format

FUNCTION SIGN (argument-1)

Arguments

Argument-1 shall be class numeric.

Returned Values

The equivalent arithmetic expression shall be as follows:

1. When the value of argument-1 is positive,
   (1)
2. When the value of argument-1 is zero,
   (0)
3. When the value of argument-1 is negative,
   (-1)

Sample

```
display "FUNCTION SIGN(numeric):" upon sysout
display FUNCTION SIGN(123) upon sysout
display FUNCTION SIGN(-123) upon sysout
display FUNCTION SIGN(0) upon sysout
display FUNCTION SIGN(4535.2) upon sysout
display FUNCTION SIGN(-348934.43) upon sysout
```

Sample Output

```
FUNCTION SIGN(numeric):
1
-1
0
1
-1
```

SIN Function

The SIN function returns a numeric value that approximates the sine of an angle or arc, expressed in radians, that is specified by argument-1.

The type of this function is numeric.

General format

FUNCTION SIN (argument-1)

Arguments

Argument-1 shall be class numeric.

Returned values

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.

Sample

```
display "FUNCTION SIN(numeric):" upon sysout
display FUNCTION SIN(1000) upon sysout
display FUNCTION SIN(100) upon sysout
display FUNCTION SIN(1) upon sysout
display FUNCTION SIN(.123) upon sysout
display FUNCTION SIN(0) upon sysout
display FUNCTION SIN(-.123) upon sysout
display FUNCTION SIN(-1) upon sysout
display FUNCTION SIN(-100) upon sysout
display FUNCTION SIN(-1000) upon sysout
```

Sample Output

```
FUNCTION SIN(numeric):
0.8268795405320025
-0.5063656411097588
0.8414709848078965
0.12269009002431533
0.0
-0.12269009002431533
```

```
-0.8414709848078965
0.5063656411097588
-0.8268795405320025
```

SOUNDEX function

The SOUNDEX function applies the surname function SOUNDEX over a surname, giving a one to four-character code which may be used to match against other SOUNDEX codes. SOUNDEX is useful for matching a surname when the exact spelling is not known; similar sounding names are given identical codes. This function may be properly used to generate a lookup key when the proper spelling is not known for the query.

General format

FUNCTION SOUNDEX (argument-1)

Arguments

Argument-1 shall be class alphanumeric.

Returned-values

The returned value is the soundex code for the given surname.

Sample Program

```
identification division.
program-id. test_soundex.

procedure division.
main-paragraph.

    display "" upon sysout
    display "FUNCTION SOUNDEX ('JONES'):" upon sysout
    display function soundex ('JONES') upon sysout

    display "FUNCTION SOUNDEX ('OLIVER'):" upon sysout
    display function soundex ('OLIVER') upon sysout

    display "FUNCTION SOUNDEX ('SMITH'):" upon sysout
    display function soundex ('SMITH') upon sysout

    display "FUNCTION SOUNDEX ('WHITE'):" upon sysout
    display function soundex ('WHITE') upon sysout
    .
```

Sample Output

```
FUNCTION SOUNDEX ('JONES'):
S415
FUNCTION SOUNDEX ('OLIVER'):
S415
FUNCTION SOUNDEX ('SMITH'):
S53
FUNCTION SOUNDEX ('WHITE'):
S53
```

SQRT Function

The SQRT function returns a numeric value that approximates the square root of argument-1.

The type of this function is numeric.

General format

FUNCTION SQRT (argument-1)

Arguments

1. Argument-1 shall be class numeric.
2. The value of argument-1 shall be zero or positive.

Returned values

1. When standard arithmetic is specified, argument-1 is not rounded.
2. The returned value shall be the absolute value of the approximation of the square root of argument-1.

Sample

```
display "FUNCTION SQRT(numeric):" upon sysout
display FUNCTION SQRT(4) upon sysout
display FUNCTION SQRT(5) upon sysout
display FUNCTION SQRT(6) upon sysout
display FUNCTION SQRT(7) upon sysout
display FUNCTION SQRT(8) upon sysout
display FUNCTION SQRT(9) upon sysout
display FUNCTION SQRT(10) upon sysout
display FUNCTION SQRT(16) upon sysout
display FUNCTION SQRT(25) upon sysout
display FUNCTION SQRT(36) upon sysout
display FUNCTION SQRT(49) upon sysout
display FUNCTION SQRT(50) upon sysout
```

Sample Output

```
FUNCTION SQRT(numeric):
2.0
2.23606797749979
2.449489742783178
2.6457513110645907
2.8284271247461903
3.0
3.1622776601683795
4.0
5.0
6.0
7.0
7.0710678118654755
```

STANDARD-COMPARE Function

The STANDARD-COMPARE function returns a character indicating the result of comparing argument-1 and argument-2 using the ordering specified in ISO/IEC 14651.

The function type is alphanumeric.

General format

FUNCTION STANDARD-COMPARE (argument-1 argument-2 [argument-3])

Arguments

1. Argument-1 shall be of class alphabetic, alphanumeric, or national.
2. Argument-2 shall be of class alphabetic, alphanumeric, or national.
3. Argument-1 and argument-2 may be of different classes.
4. Argument-3 shall be an integer whose value is in the range 0 to 3, inclusive.
5. Argument-3 specifies a level of ISO/IEC 14651 precision to be used in the comparison.
6. If argument-3 is unspecified, level 3 is implied.

Returned values

1. If ISO/IEC 14651 is not supported on the processor, or the specified level is not supported, the EC-ORDER-NOT-SUPPORTED exception condition exists.
2. If the arguments are of different classes, and one is national, the other argument is converted to class national for purposes of comparison.
3. Argument-1 and argument-2 may be of unequal lengths.
4. Argument-1 and argument-2 are compared in accordance with the ordering specified in ISO/IEC 14651 at the level of comparison indicated by argument-3.

   NOTE - The result of this function may vary for different local tailoring of the ordering defined in ISO/IEC 14651.

5. The returned value is:

"="	if the arguments compare equal
"<"	if argument-1 is less than argument-2
">"	if argument-1 is greater than argument-2

6. The length of the returned value is 1.

Sample

```
display "FUNCTION STANDARD-COMPARE(arg-1 arg-2):" upon sysout
display FUNCTION STANDARD-COMPARE("A" "B") upon sysout
display FUNCTION STANDARD-COMPARE("B" "A") upon sysout
display FUNCTION STANDARD-COMPARE("A" "A") upon sysout
```

Sample Output

```
FUNCTION STANDARD-COMPARE(arg-1 arg-2):
<
```

>
=

STANDARD-DEVIATION Function

The STANDARD-DEVIATION function returns a numeric value that approximates the standard deviation of its arguments.

The type of this function is numeric.

General format

FUNCTION STANDARD-DEVIATION ({argument-1} …)

Arguments

Argument-1 shall be class numeric.

Returned values

The equivalent arithmetic expression shall be as follows:

FUNCTION SQRT (FUNCTION VARIANCE (argument-1))

where the arguments for the VARIANCE function are the same as the arguments for the STANDARD-DEVIATION function itself.

Sample

```
display "FUNCTION STANDARD-DEVIATION(...):" upon sysout
display FUNCTION STANDARD-DEVIATION( 1 2 3 4 ) upon sysout
display FUNCTION STANDARD-DEVIATION( 4 3 2 1 ) upon sysout
display FUNCTION STANDARD-DEVIATION( 123.456 123.231 123.855 123.323 ) upon
sysout
```

Sample Output

```
FUNCTION STANDARD-DEVIATION(...):
1.118033988749895
1.118033988749895
0.23827229696294952
```

SUBTRACT-DURATION Function

The SUBTRACT-DURATION function subtracts a duration from a date, time or timestamp item, returning a new corresponding item.

General format

FUNCTION SUBTRACT-DURATION (argument-1 {date-time-keyword argument-3})

Where date-time-keyword is one of:

YEARS

MONTHS

DAYS

HOURS

MINUTES

SECONDS

MICROSECONDS

Arguments

Argument-1 is a date-time item.

Argument-3 is units of date-time-keyword. At least one date-time-keyword argument-3 pair must be specified, but each date-time-keyword may be specified no more than once.

Returned values

The date-time of argument-1 plus the duration specified.

Sample

```
id division.
program-id. subtract.

data division.
working-storage section.

  01 date-group.
*                    "@Y-%m-%d"
     05 plain-date format of date.

*                    "%H:%M:%S"
  01 time-group.
     05 plain-time format of time.

*                    "@Y-%m-%d-%H.%M.%S.@Sm"
  01 timestamp-group.
     05 plain-timestamp format of timestamp.
      05 other-timestamp format of timestamp.

  01 packed-decimal-item pic s9(9)v9(9).

  01 numedited pic $$$$,$$$.99.

  procedure division.
  main-paragraph.

     move "1972-08-06" to plain-date
     move "12:34:56" to plain-time
     move "1972-08-06-12.34.56.123456" to plain-timestamp
      move "1972-09-08-16.11.22.333333" to other-timestamp
      .

  SUBTRACT-DURATION.

    display "" upon sysout
      display "SUBTRACT-DURATION:" upon sysout
      display "" upon sysout

    display plain-date
            upon sysout
     assert plain-date = "1972-08-06"
```

```
display function subtract-duration(plain-date years 1)
        upon sysout

   assert function subtract-duration(plain-date years 1)=
      "1971-08-06"

display function subtract-duration(plain-date months 1)
        upon sysout
   assert function subtract-duration(plain-date months 1)=
      "1972-07-06"

display function subtract-duration(plain-date days 1)
        upon sysout
   assert function subtract-duration(plain-date days 1)=
      "1972-08-05"

display function subtract-duration(
        plain-date
        years 1
        months 1
        days 1
        )
        upon sysout

assert function subtract-duration(
        plain-date
        years 1
        months 1
        days 1
        )
      ="1971-07-05"

display plain-time
        upon sysout
   assert plain-time = "12:34:56"

display function subtract-duration(plain-time hours 1)
        upon sysout
   assert function subtract-duration(plain-time hours 1)=
      "11:34:56"
display function subtract-duration(plain-time minutes 1)
        upon sysout
   assert function subtract-duration(plain-time minutes 1)=
      "12:33:56"

display function subtract-duration(plain-time seconds 1)
        upon sysout
   assert function subtract-duration(plain-time seconds 1)=
      "12:34:55"

display function subtract-duration(
        plain-time
        hours 1
        minutes 1
        seconds 1
        )
```

```
          upon sysout
   assert function subtract-duration(
          plain-time
          hours 1
          minutes 1
          seconds 1
          )
        ="11:33:55"

display plain-timestamp
          upon sysout

   assert plain-timestamp = "1972-08-06-12.34.56.123456"

display function subtract-duration(plain-timestamp years 1)
          upon sysout

   assert function subtract-duration(plain-timestamp years 1)=
        "1971-08-06-12.34.56.123456"

display function subtract-duration(plain-timestamp months 1)
          upon sysout
   assert function subtract-duration(plain-timestamp months 1)=
        "1972-07-06-12.34.56.123456"

display function subtract-duration(plain-timestamp days 1)
          upon sysout
   assert function subtract-duration(plain-timestamp days 1)=
        "1972-08-05-12.34.56.123456"

display function subtract-duration(plain-timestamp hours 1)
          upon sysout
   assert function subtract-duration(plain-timestamp hours 1)=
        "1972-08-06-11.34.56.123456"

display function subtract-duration(plain-timestamp minutes 1)
          upon sysout
   assert function subtract-duration(plain-timestamp minutes 1)=
        "1972-08-06-12.33.56.123456"

display function subtract-duration(plain-timestamp seconds 1)
          upon sysout
   assert function subtract-duration(plain-timestamp seconds 1)=
        "1972-08-06-12.34.55.123456"

display function subtract-duration(
        plain-timestamp
        microseconds 1
        )
          upon sysout
   assert function subtract-duration(plain-timestamp
        microseconds 1)
        ="1972-08-06-12.34.56.123455"

 display function subtract-duration(
          plain-timestamp
```

```
            years 1
            months 1
            days 1
            hours 1
            minutes 1
            seconds 1
            microseconds 1
            )
            upon sysout

        assert function subtract-duration(
            plain-timestamp
            years 1
            months 1
            days 1
            hours 1
            minutes 1
            seconds 1
            microseconds 1
            )
          ="1971-07-05-11.33.55.123455"
            .
```

Sample Output

```
SUBTRACT-DURATION:
1972-08-06
1971-08-06
1972-07-06
1972-08-05
1971-07-05
12:34:56
11:34:56
12:33:56
12:34:55
11:33:55
1972-08-06-12.34.56.123456
1971-08-06-12.34.56.123456
1972-07-06-12.34.56.123456
1972-08-05-12.34.56.123456
1972-08-06-11.34.56.123456
1972-08-06-12.33.56.123456
1972-08-06-12.34.55.123456
1972-08-06-12.34.56.123455
1971-07-05-11.33.55.123455
```

SUM Function

The SUM function returns a value that is the sum of the arguments.

The type of this function depends upon the argument types as follows:

Argument type	Function type
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

General format

FUNCTION SUM ({argument-1} ...)

Arguments

Argument-1 shall be class numeric.

Returned values

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-11 + argument-12)
3. For n occurrences of argument-1,
   (argument-11 + argument-12 + ... + argument-1n)
   a. If the argument-1 series is all integers, the value returned is an integer.
   b. If the argument-1 series is not all integers, a numeric value is returned.

Sample

```
display "FUNCTION SUM(...):" upon sysout
display FUNCTION SUM( 1 2 3 4 ) upon sysout
display FUNCTION SUM( 4 3 2 1 ) upon sysout
display FUNCTION SUM( 123.456 123.231 123.855 123.323 ) upon sysout
```

Sample Output

```
FUNCTION SUM(...):
10
10
493.865
```

TAN Function

The TAN function returns a numeric value that approximates the tangent of an angle or arc, expressed in radians, that is specified by argument-1.

The type of this function is numeric.

General format

FUNCTION TAN (argument-1)

Arguments

Argument-1 shall be class numeric.

Returned values

The returned value is the approximation of the tangent of argument-1.

Sample

```
display "FUNCTION TAN(numeric):" upon sysout
display FUNCTION TAN(1000) upon sysout
display FUNCTION TAN(100) upon sysout
```

```
display FUNCTION TAN(1) upon sysout
display FUNCTION TAN(.123) upon sysout
display FUNCTION TAN(0) upon sysout
display FUNCTION TAN(-.123) upon sysout
display FUNCTION TAN(-1) upon sysout
display FUNCTION TAN(-100) upon sysout
display FUNCTION TAN(-1000) upon sysout
```

Sample Output

```
FUNCTION TAN(numeric):
1.4703241557027185
-0.5872139151569291
1.5574077246549023
0.12362406586927442
0.0
-0.12362406586927442
-1.5574077246549023
0.5872139151569291
-1.4703241557027185
```

TEST-DATE-TIME Function

The TEST-DATE-TIME function tests an argument to see if it is a valid date/time. It returns 1 if the argument is a valid date/time, a 0 if the argument is not a valid date/time. Because the current implementation is flexible in its input, TEST-DATE-TIME currently always returns 1.

General format

FUNCTION TEST-DATE-TIME (argument-1 date-time-type [LOCALE argument-3])

Where date-time-type is one of the following:

DATE

TIME

TIMESTAMP

Arguments

Argument-1 is a date, time, or timestamp, or an alphanumeric or an integer.

Sample

```
id division.
program-id. testdatetime.

data division.
working-storage section.

01 date-group.
*                       "@Y-%m-%d"
  05 plain-date format of date.

*                       "%H:%M:%S"
01 time-group.
  05 plain-time format of time.
```

```
*                       "@Y-%m-%d-%H.%M.%S.@Sm"
01 timestamp-group.
   05 plain-timestamp format of timestamp.
   05 other-timestamp format of timestamp.

procedure division.
main-paragraph.
   move "1972-08-06" to plain-date
   move "12:34:56" to plain-time
   move "1972-08-06-12.34.56.123456" to plain-timestamp
   move "1972-09-08-16.11.22.333333" to other-timestamp
        .

   TEST-DATE-TIME.
      display "" upon sysout
       display "TEST-DATE-TIME:" upon sysout
       display "" upon sysout

       display function test-date-time
          (plain-timestamp)
          upon sysout
       display function test-date-time
          (plain-timestamp timestamp)
          upon sysout
       display function test-date-time
          (plain-timestamp timestamp "@Y-%m-%d-%H.%M.%S.@Sm")
          upon sysout
       display function test-date-time
          (plain-timestamp timestamp locale)
          upon sysout
          .

       display function test-date-time
          (plain-time)
          upon sysout
       display function test-date-time
          (plain-time time)
          upon sysout
       display function test-date-time
          (plain-time time locale)
          upon sysout
       display function test-date-time
          (plain-time time "%H:%M:%S")
          upon sysout
          .

       display function test-date-time
          (plain-date)
          upon sysout
       display function test-date-time
          (plain-date date)
          upon sysout
       display function test-date-time
          (plain-date date locale)
          upon sysout
```

```
display function test-date-time
   (plain-date date "@Y-%m-%d")
   upon sysout
   .
```

Sample Output

```
TEST-DATE-TIME:
1
1
1
1
1
1
1
1
1
1
1
1
```

UPPER-CASE Function

The UPPER-CASE function returns a character string that is the same length as argument-1 with each lowercase letter replaced by the corresponding uppercase letter.

The type of the function depends on the argument type as follows:

Argument type	Function type
Alphanumeric	Alphanumeric
National	National

General format

FUNCTION UPPER-CASE (argument-1)

Arguments

Argument-1 shall be of class alphabetic, alphanumeric, or national and shall be at least one character position in length.

Returned values

1. The same character string as argument-1 is returned, except that each lowercase letter is replaced by the corresponding uppercase letter.
2. The character string returned has the same length as argument-1.

Sample

```
display "FUNCTION UPPER-CASE(alphanumeric):" upon sysout
display FUNCTION UPPER-CASE("UPPER CASE") upon sysout
display FUNCTION UPPER-CASE("lower case") upon sysout
display FUNCTION UPPER-CASE("MiXeD CaSe") upon sysout
```

Sample Output

```
FUNCTION UPPER-CASE(alphanumeric):
UPPER CASE
LOWER CASE
```

MIXED CASE

URL-DECODE Function

The URL-DECODE function decodes a string encoded in URL format. URL format is the encoding used for webpage addresses and numerous other internet-related activities. It ensures that only a limited character set is used to represent all characters; special characters which could have additional meanings are safely escaped into a plain text form. A piece of text which is URL encoded will generally be longer than a piece of text which is not URL encoded.

General format

FUNCTION URL-DECODE (argument-1)

Arguments

Argument-1 shall be alphanumeric.

Returned Values

The returned value is the URL decoded form of argument-1.

Sample Program

```
identification division.
program-id. sample_url.

data division.
working-storage section.

01 my-item pic x(100).
01 my-item-2 pic x(100).

procedure division.
main-paragraph.
   move function url-encode( "Hi there everybody! How are you?" ) to my-item
   display "Encoded:" my-item upon sysout

   move function url-decode( my-item ) to my-item-2
   display "Decoded:" my-item-2 upon sysout
   .
```

Sample Output

```
Encoded:Hi+there+everybody%21+How+are+you%3F
Decoded:Hi there everybody! How are you?
```

URL-ENCODE Function

The URL-ENCODE function decodes a string encoded in URL format. URL format is the encoding used for web page addresses and numerous other internet-related activities. It ensures that only a limited character set is used to represent all characters; special characters which could have additional meanings are safely escaped into a plain text form. A piece of text which is URL encoded will generally be longer than a piece of text which is not URL encoded.

General format

FUNCTION URL-ENCODE (argument-1)

Arguments

Argument-1 shall be alphanumeric.

Returned Values

The returned value is the URL encoded form of argument-1.

Sample Program

```
id division.
program-id. sample_url.

data division.
working-storage section.

01 my-item pic x(100).
01 my-item-2 pic x(100).

procedure division.
main-paragraph.
   move function url-encode( "Hi there everybody! How are you?" ) to my-item
   display "Encoded:" my-item upon sysout

   move function url-decode( my-item ) to my-item-2
   display "Decoded:" my-item-2 upon sysout
   .
```

Sample Output

```
Encoded:Hi+there+everybody%21+How+are+you%3F
Decoded:Hi there everybody! How are you?
```

VARIANCE Function

The VARIANCE function returns a numeric value that approximates the variance of its arguments.

The type of this function is numeric.

General format

FUNCTION VARIANCE ({argument-1} ...)

Arguments

Argument-1 shall be class numeric.

Returned values

1. The equivalent arithmetic expression shall be as follows:
   a. For one occurrence of argument-1,

      (0)

   b. For two occurrences of argument-1,

      (((argument-1$_1$ - FUNCTION MEAN (argument-1)) ** 2 +

(argument-1$_2$- FUNCTION MEAN (argument-1)) ** 2) / 2)

c. 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)

where the argument of the MEAN function and argument-1i of the SUM function are the same as the arguments for the VARIANCE function itself.

Sample

```
display "" upon sysout
display "FUNCTION VARIANCE(...):" upon sysout
display FUNCTION VARIANCE( 1 2 3 4 ) upon sysout
display FUNCTION VARIANCE( 4 3 2 1 ) upon sysout
display FUNCTION VARIANCE( 123.456 123.231 123.855 123.323 ) upon sysout
```

Sample Output

```
FUNCTION VARIANCE(...):
1.25
1.25
0.0567736875
```

WHEN-COMPILED Function

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 type of this function is alphanumeric.

General format

FUNCTION WHEN-COMPILED

Returned values

The character positions returned, numbered from left to right, are:

Character Positions	Contents
1-4	Four numeric digits of the year in the Gregorian calendar.
5-6	Two numeric digits of the month of the year, in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range 00 through 23.
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric digits of the seconds past the minute, in the range 00 through 59.
15-16	Two numeric digits of the hundredths of a second past the second, in the range 00 through 99. The value 00 is returned if the system on which the function is evaluated does not have the facility to provide the fractional part of a second.
17	Either the character '-', the character '+', or the character '0'. The character '-' is returned if the local time indicated in the

	previous character positions is behind Greenwich Mean Time. The character '+' is returned if the local time indicated is the same as or ahead of Greenwich Mean Time. The character '0' is returned if the system on which this function is evaluated does not have the facility to provide the local time differential factor.
18-19	If character position 17 is '-', two numeric digits are returned in the range 00 through 12 indicating the number of hours that the reported time is behind Greenwich Mean Time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the reported time is ahead of Greenwich Mean Time. If character position 17 is '0', the value 00 is returned.
20-21	Two numeric digits are returned in the range 00 through 59 indicating the number of additional minutes that the reported time is ahead of or behind Greenwich Mean Time, depending on whether character position 17 is '+' or '-', respectively. If character position 17 is '0', the value 00 is returned.

1. The returned value is the date and time of compilation of the source program that contains this function. If the program is a contained program, the returned value is the compilation date and time associated with the separately-compiled program in which it is contained.
2. The returned value shall denote the same time as the compilation date and time if provided in the listing of the source program and in the generated object code for the source program, although their representations and precision may differ.

Sample

```
display "FUNCTION WHEN-COMPILED:" upon sysout
display FUNCTION WHEN-COMPILED upon sysout

FUNCTION WHEN-COMPILED:
2001020614204700000000
```

YEAR-TO-YYYY Function

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.

General format

FUNCTION YEAR-TO-YYYY (argument-1 [argument-2])

Arguments

1. Argument-1 shall be a nonnegative integer that is less than 100.
2. Argument-2 shall be an integer.
3. If argument-2 is omitted, the function shall be evaluated as though 50 were specified.

4. The sum of the year at the time of execution and the value of argument-2 shall be less than 10000 and greater than 1699.

Returned values

1. Maximum-year shall be calculated as follows:

   (FUNCTION NUMVAL (FUNCTION CURRENT-DATE (1:4)) + argument-

   where argument-2 of the NUMVAL function is the same as argument-2 of the YEAR-TO-YYYY function itself.

2. The equivalent arithmetic expression shall be as follows:

   a. When the following condition is true

      (FUNCTION MOD (maximum-year, 100) >= argument-1)

      The equivalent arithmetic expression shall be

      (argument-1 + 100 * (FUNCTION INTEGER (maximum-year/100)))

   b. Otherwise, the equivalent arithmetic expression shall be

      (argument-1 + 100 * (FUNCTION INTEGER (maximum-year/100) - 1))

NOTES

1. 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.
2. 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:

   (fixed-maximum-year - FUNCTION NUMVAL (FUNCTION CURRENT-DATE (1:4)))

   If the fixed window is 1973 through 2072, then in 2009 argument-2 shall have the value of 63 and in 2019, the value of 53.

Sample

```
display "FUNCTION YEAR-TO-YYYY(YYMMDD):" upon sysout
display FUNCTION YEAR-TO-YYYY(76 50) upon sysout
display FUNCTION YEAR-TO-YYYY(69 50) upon sysout
display FUNCTION YEAR-TO-YYYY(71 50) upon sysout
display FUNCTION YEAR-TO-YYYY(72 50) upon sysout
display FUNCTION YEAR-TO-YYYY(00 50) upon sysout
display FUNCTION YEAR-TO-YYYY(01 50) upon sysout
display FUNCTION YEAR-TO-YYYY(23 50) upon sysout
```

Sample Output

```
FUNCTION YEAR-TO-YYYY(YYMMDD):
1976
1969
1971
1972
2000
2001
2023
```

16. Standard classes

Java classes and objects are standard with Elastic COBOL. No extra steps are necessary to use the functionality present with the Java implementation running a Elastic COBOL program.

Index7

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z