isCOBOL Evolve: Language Reference

Key Topics

- Program Structure
- File Names - File names interpretation
- Data Description
- Procedure Division Statements
- Embedded SQL Statements
- Object-oriented Programming
- EFD Directives

Table of Contents

Preface

This document describes the isCOBOL Evolve syntax and functions and is based on and extends the ANSI® X3.23-1985 revision of ANSI X3.23-1974 and the ISO/IEC 1989:200x WD 1.5 document.

Acknowledgement

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 COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

The authors and copyright holders of the copyrighted materials used herein

FLOW-MATIC (trademark of Sperry Rand Corporation). Programming for the UNIVAC® I and II, Data Automation Systems copyrighted 1958,1959, by Sperry Rand Corporation; IBM Commercial Translater Form No. F 28-8013, copyrighted 1959 by IBM: FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

ITI/INCITS claim a copyright interest in the COBOL standard to the extent that these organizations have verified that the COBOL standard has met the standard setting requirements of these organizations.

However, ITI/INCITS claims no copyright in the underlying technical specifications of the COBOL standard and, to the extent that consent is required, INCITS freely consents to use of the underlying technical specification by any person without charge. No person reproducing, improving, or using the standard shall claim that any modifications or improvements have been adopted as part of any "official" COBOL standards and/or the standard setting processes of ANSI/ITI/INCITS.

Copyright Notice

Definitions

Alphabet-Name

A User-defined word, in the SPECIAL-NAMES Paragraph of the Environment Division, that assigns a name to a Collating Sequence.

Alphabetic Character

A Letter or a space Character.

Alphanumeric Character

Any Character in the computer character set.

Character

The basic indivisible unit of the language.

Character Position

A character position is the amount of physical storage required to store a single standard data format character whose usage is DISPLAY. Further characteristics of the physical storage are defined by the implementor.

Character-String

A sequence of contiguous Characters which form a COBOL Word, a Literal, a PICTURE character-string, or a Comment-Entry.

COBOL Word

A Character-String which forms a User-defined word, a System name, or a Reserved Word.

Collating Sequence

The sequence in which the characters that are acceptable to a computer are ordered for purposes of sorting, merging, comparing, and for processing indexed files sequentially.

Comment-Entry

An entry in the Identification Division that may be any combination of Characters from the computer character set.

Data Item

A unit of data (excluding Literals) defined by the COBOL program.

Figurative Constant

Any of the following literals:

Literal	Meaning
ALL	Represents all or part of the string generated by successive concatenations of the characters comprising the literal. It must be a Nonnumeric Literal.
HIGH-VALUE, HIGH-VALUES	Represents one or more characters with the highest ordinal position in the program collating sequence. Usually this is the hexadecimal value "FF".

Literal	Meaning
LOW-VALUE, LOW-VALUES	Represents one or more characters with the lowest ordinal position in the program collating sequence. Usually this is the binary value 0.
NULL, NULLS	Represents the numeric value "zero" or one or more occurrences of a character whose underlying representation is binary zero. It also represents an invalid object reference when it is used in conjunction with OBJECT REFERENCE data types.
QUOTE, QUOTES	Represents one or more quotation mark characters. These words may not be used in place of quotation marks for delimiting a Nonnumeric Literal.
SPACE, SPACES	Represents one or more space characters.
ZERO, ZEROS, ZEROES	Represents the numeric value "zero" or one or more occurrences of the character 0, depending on whether the constant is treated as a numeric or nonnumeric literal.

Identifier

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

Integer

A Numeric Literal or a numeric data item that does not include any digit position to the right of the assumed decimal point. When the term 'integer' appears in general formats, integer must not be a numeric data item, and must not be signed, nor zero unless explicitly allowed by the rules of that format.

Letter

A Character belonging to one of the following sets:

1. Uppercase letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z;
2. Lowercase letters: a, b, c, d, e, f, g, h, i, j, k, 1, m, n, o, p, q, r, s, t, u, v, w, x, y, z
3. Accented letters: è, é, à, ù, ò, ì.
4. Other alphabets, according with the current encoding.

Literal

A Character-String whose value is implied by the ordered set of characters comprising the string.

DBCS Literal

A double-byte Character-String, a character set that uses two-byte (16-bit) characters rather than one-byte (8-bit) characters.

Nonnumeric Literal

- A Literal bound by quotation marks or apostrophes. The beginning and ending delimiters must be the same (that is, either both quotes or both apostrophes). The string of characters may include any Character in the computer's character set. To place the delimiter character in a nonnumeric literal, use two contiguous delimiter characters (either two quotes or two apostrophes). These two characters represent a single occurrence of that character.
- The hexadecimal value of one or more characters using the native character set. Any of the following formats are recognized:
    - X"hex-values"

- X'hex-values'
- H"hex-values"
- H'hex-values'
- N"national-digits"
- N’national-digits’
- NX"national-hex-digits"
- NX’national-hex-digits’

NOTE - National digits are managed as UTF-16BE characters.

- The name of a resource property specified by any of the following syntaxes:
  - R"property_name"
  - R'property_name'

Numeric Character

A Character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Numeric Data Item

A Data Item whose description restricts its content to a value represented by characters chosen from the digits '0' through '9'; if signed, the item may also contain a '+', '-', or other representation of an operational sign. (See the SIGN clause.)

Numeric Literal

A Literal composed of one or more Numeric Characters that may contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

Numeric literals may also be specified using binary, octal, or hexadecimal notation. To specify a numeric literal in one of these forms, preface the number with one of the following prefixes:

Prefix	Notation	Usage Example (representation of zero)
B#	Binary	B#110000
O#	Octal	O#60
X#	Hexadecimal	X#30
H#	Hexadecimal	H#30

Reserved Word

A COBOL Word specified in the list of words which may be used in a COBOL source program, but which must not appear in the program as a User-defined word or System name.

System name

A COBOL Word that is used to communicate with the operating environment.

User-defined word

A user-defined word is a COBOL Word that must be supplied by the user to satisfy the format of a clause or statement. Each Character of a user-defined word is selected from Literal, Numeric Literal, '_' and '-', except that '_' and '-' may not appear as the first character.

Concatenation Expressions

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

```
literal-1 & literal-2
```

Both operands must be of the same class, except that a figurative constant may be specified as one or both operands. Neither operand must be numeric. Neither literal-1 nor literal-2 must be a figurative constant that begins with the word ALL.

The value of a concatenation expression is the concatenation of values of the literals, figurative constants and concatenation expressions of which it is composed and may be used anywhere a literal of that class may be used.

Source Formats

isCOBOL supports four types of source format:

- Fixed
- Free
- Terminal
- Variable

Fixed

The Fixed format (also known as the Ansi format) divides the COBOL source row into 72 columns, that are used in the following way:

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

- Division headers, section headers and paragraph names must start in Area A.
- Level numbers can appear either in Area A or Area B.
- All other COBOL text must start in Area B.
- Comments are identified by an asterisk, a slash or a dollar sign in the Indicator area (in this case the whole row is commented) or by a pipe in Area B (in this case only the text after the pipe is commented).
- Conditional debugging lines are identified by the "D" character in the Indicator area. These lines are treated as comment unless WITH DEBUGGING MODE is specified in the SOURCE-COMPUTER Paragraph.
- All text before column 7 and after column 72 is ignored by the Compiler.

Note - this source format is affected by the -sl compiler option and by the IMP MARGIN-R IS AFTER END OF RECORD Directive. When one of them is used, Area B is no longer limited to column 72; it has no limit.

Line continuation

Sentences, entries, phrases, and clauses that continue in Area B of subsequent lines are called continuation lines. A hyphen in a line's indicator area causes the first nonblank character in Area B to be the immediate successor of the last nonblank character of the preceding line. This continuation excludes intervening comment lines and blank lines.

If the continued line ends with a nonnumeric literal without a closing quotation mark, the first nonblank character in Area B of the continuation line must be a quotation mark. The continuation starts immediately after the quotation mark. All spaces at the end of the continued line are part of the literal.

If the indicator area of the continuation line is blank, the compiler treats the last nonblank character of the preceding line as if it were followed by a space.

If the indicator area of the continuation line is blank, but the rest of the line is not blank, the compiler considers the current literal as closed and raises the informational error #287.

Free

The Free format divides the COBOL source row into 512 columns that can be freely used. There are no areas in which to store the code, so the text can appear anywhere within the row.

Comments are identified by the “*>” characters and can appear anywhere within the row.

Terminal

The Terminal format divides the COBOL source row into 512 columns, that are used in the following way:

Column 1	Indicator area
Columns 1 to 4	Area A
Columns 5 to 512	Area B

- Division headers, section headers and paragraph names must start in Area A.
- Level numbers can appear either in Area A or Area B.
- All other COBOL text must start in Area B.
- Comments are identified by an asterisk, a slash or a dollar sign in the Indicator area (in this case the whole row is commented) or by a pipe in Area B (in this case only the text after the pipe is commented).
- Conditional debugging lines are identified by the "\D" characters sequence in the Indicator area. These lines are treated as comment unless WITH DEBUGGING MODE is specified in the SOURCE-COMPUTER Paragraph.

Line continuation

Sentences, entries, phrases, and clauses that continue in Area B of subsequent lines are called continuation lines. A hyphen in a line's indicator area causes the first nonblank character in Area B to be the immediate successor of the last nonblank character of the preceding line. This continuation excludes intervening comment lines and blank lines.

If the continued line ends with a nonnumeric literal without a closing quotation mark, the first nonblank character in Area B of the continuation line must be a quotation mark. The continuation starts immediately after the quotation mark. All spaces at the end of the continued line are part of the literal.

If the indicator area of the continuation line is blank, the compiler treats the last nonblank character of the preceding line as if it were followed by a space.

If the indicator area of the continuation line is blank, but the rest of the line is not blank, the compiler considers the current literal as closed and raises the informational error #287.

Variable

The Variable format divides the COBOL source row into areas as in Fixed format. The only difference from Fixed format is that the B area has no fixed right margin, though for practical purposes this implementation restricts the maximum length of a source line to 250 single-byte or 125 double-byte characters:

Columns 1 to 6	Sequence number
Column 7	Indicator area
Columns 8 to 11	Area A
Coulmns 12 to 250	Area B

- Division headers, section headers and paragraph names must start in Area A.
- Level numbers can appear either in Area A or Area B.
- All other COBOL text must start in Area B.
- Comments are identified by an asterisk, a slash or a dollar sign in the Indicator area (in this case the whole row is commented) or by a pipe in Area B (in this case only the text after the pipe is commented). Putting a star in column 1 creates a comment that will not be included in the list.
- Conditional debugging lines are identified by the "D" character in the Indicator area. These lines are treated as comment unless WITH DEBUGGING MODE is specified in the SOURCE-COMPUTER Paragraph.
- All text before column 7 and after column 250 is ignored by the Compiler.

Line continuation

Sentences, entries, phrases, and clauses that continue in Area B of subsequent lines are called continuation lines. A hyphen in a line's indicator area causes the first nonblank character in Area B to be the immediate successor of the last nonblank character of the preceding line. This continuation excludes intervening comment lines and blank lines.

If the continued line ends with a nonnumeric literal without a closing quotation mark, the first nonblank character in Area B of the continuation line must be a quotation mark. The continuation starts immediately after the quotation mark. All spaces at the end of the continued line are part of the literal.

If the indicator area of the continuation line is blank, the compiler treats the last nonblank character of the preceding line as if it were followed by a space.

If the indicator area of the continuation line is blank, but the rest of the line is not blank, the compiler considers the current literal as closed and raises the informational error #287.

Chapter 1

Program Structure

Synopsis

The schema below shows how a COBOL program is structured. For simplifying access to information, it has been split into smaller parts.

```
[{IDENTIFICATION} DIVISION. ]
 {ID             }

 { {PROGRAM-ID } . Program-Name [ IS {INITIAL } PROGRAM ] .
   {FUNCTION-ID}                    {RESIDENT}

     [ AUTHOR. [Comment-Entry .] ]

     [ INSTALLATION. [Comment-Entry .] ]

     [ DATE-WRITTEN. [Comment-Entry .] ]

     [ DATE-COMPILED. [Comment-Entry .] ]

     [ SECURITY. [Comment-Entry .] ]

     [ REMARKS. [Comment-Entry .] ]

     [ ENVIRONMENT DIVISION. ]

     [ DATA DIVISION. ]

     PROCEDURE DIVISION

     [ END PROGRAM [ Program-Name ]. ] }


 { CLASS-ID. Class-Name [ AS Literal ]

                          [ INHERITS FROM Class-Name ]

                          [ IMPLEMENTS { Interface-Name } ... ] .

     [ AUTHOR. Comment-Entry . ]

     [ INSTALLATION. Comment-Entry . ]
```

```
[ DATE-WRITTEN. Comment-Entry . ]

[ DATE-COMPILED. Comment-Entry . ]

[ AUTHOR. Comment-Entry . ]

[ INSTALLATION. Comment-Entry . ]

[ DATE-WRITTEN. Comment-Entry . ]

[ DATE-COMPILED. Comment-Entry . ]

[ SECURITY. Comment-Entry . ]

[ REMARKS. Comment-Entry . ]

[ ENVIRONMENT DIVISION. ]

[ {IDENTIFICATION} DIVISION.
  {ID            }

  FACTORY.

    [ ENVIRONMENT DIVISION. ]

    [ DATA DIVISION. ]

    [ PROCEDURE DIVISION.

      [ {IDENTIFICATION} DIVISION.
        {ID            }

        METHOD-ID. Method-Name [ AS Literal ] [ IS {PUBLIC   } ] [ OVERRIDE ].
                                                  {PRIVATE  }
                                                  {PROTECTED}

          [ ENVIRONMENT DIVISION. ]

          [ DATA DIVISION. ]

          PROCEDURE DIVISION

          END METHOD [ Method-Name ]. ] ... ]

    END FACTORY. ]

[ {IDENTIFICATION} DIVISION.
  {ID            }

  OBJECT.

    [ ENVIRONMENT DIVISION. ]

    [ DATA DIVISION. ]
```

```
        [ PROCEDURE DIVISION.

          [ {IDENTIFICATION} DIVISION.
            {ID            }

            METHOD-ID. Method-Name [ AS Literal ] [ IS {PUBLIC   } ] [ OVERRIDE ].
                                                       {PRIVATE  }
                                                       {PROTECTED}

              [ ENVIRONMENT DIVISION. ]

              [ DATA DIVISION. ]

              PROCEDURE DIVISION

              END METHOD [ Method-Name ]. ] ... ]

          END OBJECT. ]
    [ END CLASS [ Class-Name ]. ] }
```

Syntax rules

1. *Program-Name* is a User-defined word, as defined in the Definitions section in the Preface of this document.
2. Each reference to *Comment-Entry* is a Comment-Entry, as defined in the Definitions section in the Preface of this document.
3. Each reference to *Class-Name* is a User-defined word, as defined in the Definitions section in the Preface of this document.
4. *Interface-Name* is a User-defined word, as defined in the Definitions section in the Preface of this document.
5. Each reference to *Method-Name* is a User-defined word, as defined in the Definitions section in the Preface of this document.
6. Each reference to *Literal* is a Nonnumeric Literal, as defined in the Definitions section in the Preface of this document.

General rules

1. PROGRAM-ID (or its synonymous FUNCTION-ID) and CLASS-ID cannot appear together in the same program.

Chapter 2

Identification Division

The IDENTIFICATION DIVISION identifies the program and is required in a COBOL source program.

Paragraph headers identify the type of information contained in the paragraph.

The name of the program must be given in the first paragraph, which can be the PROGRAM-ID Paragraph or the CLASS-ID Paragraph when defining a class.

General format

```
[{IDENTIFICATION} DIVISION. ]
 {ID             }

 {PROGRAM-ID Paragraph}
 {CLASS-ID Paragraph  }
 {INTERFACE-ID Paragraph  }
```

PROGRAM-ID Paragraph

The PROGRAM-ID paragraph specifies the name by which the program is identified.

General format

```
{PROGRAM-ID }. Program-Name [ IS {INITIAL  } PROGRAM ] .
{FUNCTION-ID}                    {RECURSIVE}
                                 {RESIDENT }

  [ AUTHOR. [ Comment-Entry-1 .] ]

  [ INSTALLATION. [ Comment-Entry-2 .] ]

  [ DATE-WRITTEN. [ Comment-Entry-3 .] ]

  [ DATE-COMPILED. [ Comment-Entry-4 .] ]

  [ SECURITY. [ Comment-Entry-5 .] ]

  [ REMARKS. [ Comment-Entry-6 .] ]

  [ Environment Division ]

  [ Data Division ]


  Procedure Division

 [ END PROGRAM [Program-Name]. ]
```

Syntax rules

1. *Program-Name* is a User-defined word, as defined in the Definitions section in the Preface of this document.
2. *Comment-Entry-1*, *Comment-Entry-2*, *Comment-Entry-3*, *Comment-Entry-4*, *Comment-Entry-5* and *Comment-Entry-6* are Comment-Entries, as defined in the Definitions section in the Preface of this document.
3. The RECURSIVE clause is treated as commentary.

General rules

1. PROGRAM-ID and FUNCTION-ID are synonymous.
2. *Program-Name* identifies the source program, the object program, and all listings pertaining to a particular program.
3. When the INITIAL clause is specified, the program is automatically cancelled from memory when it terminates. The memory used by its data items is released and all files are closed.
4. When the RESIDENT clause is specified, the program is never cancelled. CANCEL Statements have no effect on it.
5. AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, SECURITY and REMARKS are supported only for compatibility reasons and their presence does not affect the program behavior. However, they must properly formed when declared.

CLASS-ID Paragraph

The CLASS-ID paragraph specifies the name by which the class is identified.

```
CLASS-ID. Class-Name-1 [ AS Literal-1 ]
                       [ INHERITS FROM Class-Name-2 ]
                       [ IMPLEMENTS { Interface-Name } ... ] .
  [ AUTHOR. [ Comment-Entry-1 .] ]
  [ INSTALLATION. [ Comment-Entry-2 .] ]
  [ DATE-WRITTEN. [ Comment-Entry-3 .] ]
  [ DATE-COMPILED. [ Comment-Entry-4 .] ]
  [ SECURITY. [ Comment-Entry-5 .] ]
  [ REMARKS. [ Comment-Entry-6 .] ]
  [ Environment Division ]
  [ FACTORY Paragraph ]
  [ OBJECT Paragraph ]
[ END CLASS [ Class-Name-3 ] . ]
```

Syntax rules

1. *Class-Name-1*, *Class-Name-2*, *Class-Name-3* and *Interface-Name* are User-defined words, as defined in the Definitions section in the Preface of this document.
2. *Literal-1* is a Nonnumeric Literal, as defined in the Definitions section in the Preface of this document.
3. *Comment-Entry-1*, *Comment-Entry-2*, *Comment-Entry-3*, *Comment-Entry-4*, *Comment-Entry-5* and *Comment-Entry-6* are Comment-Entries, as defined in the Definitions section in the Preface of this document.

General rules

1. *Class-Name-1* is the name by which the class is identified, unless the AS clause is specified.
2. *Literal-1* is the name by which the class is identified. It may contain the name of the Java package it belongs to. If *Literal-1* is omitted, it's assumed to be the same as *Class-Name-1*.
3. *Class-Name-2* is the name of the class that *Class-Name-1* inherits from. *Class-Name-1* becomes a subclass of *Class-Name-2*. All the methods of *Class-Name-2* become available for *Class-Name-1*. If *Class-Name-1* implements a method of *Class-Name-2*, the new implementation overrides the one in *Class-Name-2*.
4. AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, SECURITY and REMARKS are supported only for compatibility reasons and their presence does not affect the program behavior. However, they must properly formed when declared.
5. *Interface-Name* must be defined in the REPOSITORY Paragraph in the Configuration Section of the ENVIRONMENT DIVISION.

INTERFACE-ID Paragraph

The INTERFACE-ID paragraph specifies the name by which the interface is identified.

```
INTERFACE-ID. Interface-Name-1 [ AS Literal-1 ]

                        [ INHERITS FROM Interface-Name-2 ]

  [ AUTHOR. [ Comment-Entry-1 .] ]

  [ INSTALLATION. [ Comment-Entry-2 .] ]

  [ DATE-WRITTEN. [ Comment-Entry-3 .] ]

  [ DATE-COMPILED. [ Comment-Entry-4 .] ]

  [ SECURITY. [ Comment-Entry-5 .] ]

  [ REMARKS. [ Comment-Entry-6 .] ]

  [ Environment Division ]

  [ FACTORY Paragraph ]

  [ OBJECT Paragraph ]

[ END INTERFACE [ Interface-Name-3 ] . ]
```

Syntax rules

1. *Interface-Name-1*, *Interface-Name-2* and *Interface-Name-3* are User-defined words, as defined in the Definitions section in the Preface of this document.
2. *Literal-1* is a Nonnumeric Literal, as defined in the Definitions section in the Preface of this document.
3. *Comment-Entry-1*, *Comment-Entry-2*, *Comment-Entry-3*, *Comment-Entry-4*, *Comment-Entry-5* and *Comment-Entry-6* are Comment-Entries, as defined in the Definitions section in the Preface of this document.

General rules

1. *Interface-name-1* names the interface declared by this interface definition.
2. *Literal-1* is the name by which the interface is identified. It may contain the name of the Java package it belongs to. If *Literal-1* is omitted, it's assumed to be the same as *Interface-Name-1*.
3. *Interface-Name-2* is the name of the class that *Interface-Name-1* inherits from. *Interface-Name-1* becomes a subclass of *Interface-Name-2*. All the methods of *Interface-Name-2* become available for *Interface-Name-1*. If *Interface-Name-1* implements a method of *Interface-Name-2*, the new implementation overrides the one in *Interface-Name-2*.
4. AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, SECURITY and REMARKS are supported only for compatibility reasons and their presence does not affect the program behavior. However, they must properly formed when declared.
5. No fields are allowed in an INTERFACE-ID.

FACTORY Paragraph

The FACTORY Paragraph introduces the factory definition.

Method-Id-Paragraph defines methods that may return an instance of the native class. Factory methods usually create new instances of private objects that are unknown to the caller. This allows for the accomplishment of several objectives, such as preventing the creation of invalid objects, automatically choosing the proper object to be instantiated, keeping track of a created object or doing some pre- or post-processing on newly created objects.

Factory methods should not rely on the status of an object instance.

General format

```
{IDENTIFICATION} DIVISION.
{ID             }

FACTORY.

  [ Environment Division ]

  [ Data Division ]

  [ PROCEDURE DIVISION.

    [ METHOD-ID Paragraph ] ... ]

END FACTORY.
```

OBJECT Paragraph

The OBJECT Paragraph introduces the object definition.

Method-Id-Paragraph defines methods that refer to a specific instance of an object. An instance of the native class must be created before using an object method.

General format

```
{IDENTIFICATION} DIVISION.
{ID             }

OBJECT.

  [ Environment Division ]

  [ Data Division ]

  Procedure Division

END OBJECT.
```

METHOD-ID Paragraph

The METHOD-ID Paragraph introduces a method definition.

General format

```
{IDENTIFICATION} DIVISION.
{ID             }

METHOD-ID.  {Method-Name [ AS Literal-1 ]} [ IS {PUBLIC    } ] [ OVERRIDE ].
            {Literal-1                   }      {PRIVATE   }
                                                {PROTECTED}
                                                {DEFAULT   }
                                                {STATIC    }

  [ Environment Division ]

  [ Data Division ]

  Procedure Division

END METHOD [{[Method-Name]}]
           {Literal-1    }.
```

Syntax rules

1. *Method-Name* is a User-defined word, as defined in the Definitions section in the Preface of this document.
2. *Literal-1* is a Nonnumeric Literal, as defined in the Definitions section in the Preface of this document.

General rules

1. *Method-Name* is the name by which the method is identified, unless the AS clause is specified. If *Method-Name* is an empty string, an unnamed method is produced. See Unnamed methods for further details.
2. *Literal-1* is the name by which the method is identified. If *Literal-1* is omitted, it's assumed to be the same as *Method-Name*.
3. When the OVERRIDE clause is specified, the method definition overrides the same method of the super class. This is the default behavior, therefore this clause is treated as a commentary.

CLASS-ID

4. In a CLASS-ID, DEFAULT and STATIC methods are not allowed.
5. When the PUBLIC phrase is specified, the method can be referenced by any program.
6. When the PRIVATE phrase is specified, the method can be referenced only by the class it refers to.
7. When the PROTECTED phrase is specified, the method can be referenced by the class it refers to and by those classes that inherit from it.
8. If neither PUBLIC nor PRIVATE nor PROTECTED are specified, the method is PUBLIC.

INTERFACE-ID

9. In an INTERFACE-ID, PUBLIC, PRIVATE and PROTECTED methods are not allowed.
10. When the DEFAULT phrase is specified,
    a. the method can have a PROCEDURE DIVISION
    b. the method must appear in the OBJECT paragraph.
11. When the STATIC phrase is specified,
    c. the method can have a PROCEDURE DIVISION
    d. the method must appear in the FACTORY paragraph.

12. When neither DEFAULT nor STATIC are specified,

    a. the method can't have a PROCEDURE DIVISION
    b. the method must appear in the OBJECT paragraph.

13. Unlike standard interface methods, DEFAULT and STATIC methods are not required to be present in the implementing class. If the implementing class doesn't include these methods, the code in the INTERFACE-ID is executed when these methods are invoked.
    DEFAULT and STATIC methods enable you to add new functionality in the interfaces without breaking the existing contract of the implementing classes.

Chapter 3

Environment Division

The Environment Division specifies those aspects of a data processing problem that may be dependent upon the physical characteristics of a specific computer.

In Object Oriented programs it may appear more than once. In that case, the definitions are valid for specific contexts, with the following rules:

1. When specified in the FACTORY Paragraph, definitions are valid for all the methods of the program.
2. When specified in the OBJECT Paragraph, definitions are valid for all the methods belonging to the OBJECT Paragraph.
3. When specified in the METHOD-ID Paragraph, definitions are valid only for that method.

General format

```
[ ENVIRONMENT DIVISION. ]

  [ CONFIGURATION Section ]

  [ INPUT-OUTPUT Section ]
```

CONFIGURATION Section

The Configuration Section contains paragraphs describing common program definitions and how the program must behave.

General format

```
CONFIGURATION SECTION.

  [ SOURCE-COMPUTER Paragraph ]

  [ OBJECT-COMPUTER Paragraph ]

  [ SPECIAL-NAMES Paragraph ]

  [ REPOSITORY Paragraph ]
```

SOURCE-COMPUTER Paragraph

The original purpose of the SOURCE-COMPUTER Paragraph was to describe the computer on which the program is compiled. Now, its only purpose is to enable the WITH DEBUGGING mode.

General format

```
SOURCE-COMPUTER.

  [ Comment [ WITH DEBUGGING MODE ].]
```

Syntax rules

1. *Comment* is a commentary entry, as defined in the Definitions section in the Preface of this document.

General rules

1. When WITH DEBUGGING MODE is specified, the "D" character in the indicator area is no longer considered as comment marker, and those lines are executed.
2. If WITH DEBUGGING MODE is specified and the program is compiled with the -cv option, the Format 3 USE statement can be used.

OBJECT-COMPUTER Paragraph

The original purpose of the OBJECT-COMPUTER Paragraph was to describe the computer on which the program is to be executed. Now, its only purpose is to alter non-numeric comparison rules for the whole program.

General format

```
OBJECT-COMPUTER.

  [ Comment .]

  [ PROGRAM COLLATING SEQUENCE IS alphabet-name ]
```

Syntax rules

1. *Comment* is a commentary entry, as defined in the Definitions section in the Preface of this document.
2. Alphabet-name is a User-defined word that describes the collating sequence of the object machine.

General rules

1. The COLLATING SEQUENCE clause specifies the collating sequence for any alphanumeric comparisons done in the program. The sequence of the characters in the alphabet determines the sequence for character comparisons. It also specifies the default collating sequence for the SORT verb.

SPECIAL-NAMES Paragraph

The SPECIAL-NAMES Paragraph contains definitions to be used by the program. The meaning of each definition is described below.

General format

```
SPECIAL-NAMES.

  [ [ Switch-Name

      [ IS Mnemonic-Name ]

      [ ON STATUS IS Condition-Name-1 ]

      [ OFF STATUS IS Condition-Name-2 ] ] ...
```

```
    [ ALPHABET Alphabet-Name-1 IS { { Literal-1 [ { {THROUGH} Literal-
2    } ] } ... } ] ...
                                                  {THRU            }
                                                { { ALSO Literal-3 } ... }
                                  { ASCII      }
                                  { EBCDIC     }
                                  { NATIVE     }
                                  { STANDARD-1 }
                                  { STANDARD-2 }

    [ CLASS Class-Name-1 IS { Literal-4 [ {THROUGH} Literal-5 } ... ] ] ...
                                        {THRU   }
    [ CONSOLE IS { CRT       } ]
                 { Literal-6 }

    [ CRT STATUS IS Crt-Status ]

    [ CURRENCY SIGN IS Literal-7 ]

    [ DECIMAL-POINT IS COMMA ]

    [ CURSOR IS cursor-name ]

    [ EVENT OBJECT IS Class-Name-2 ]

    [ EVENT SOURCE IS Object-Reference ]

    [ EVENT STATUS IS Event-Status ]

    [ SCREEN CONTROL IS Screen-Control ]

    [ SYMBOLIC CHARACTERS {Symbolic-Character} ... [{IS } ] {Literal-8} ... ]. ]
                                                   {ARE}

    [ ENVIRONMENT-NAME IS Mnemonic-Name ]

    [ ENVIRONMENT-VALUE IS Mnemonic-Name ]

    [ SYSTEM-NAME IS Mnemonic-Name ]

    [ NUMERIC SIGN IS TRAILING SEPARATE ]

    [ CSP IS Name ]

    [ C01 IS Name ]    [ C02 IS Name ]  ...  [ C12 IS Name ]

    [ SYSIN IS SystemName ]

    [ SYSOUT IS SystemName ]

    [ SYSERR IS SystemName ]

    [ SYSOUT-FLUSH IS SystemName ]

    [ { SYSLST  } IS SystemName ]
      { SYSLIST }
```

Syntax rules

1. *Switch-Name* can be one of the following:
    a. SWITCH-0, SWITCH-1, SWITCH-2 ... SWITCH-26
    b. SWITCH followed by a Letter from "A" to "Z" (bounded by quotation marks)
    c. SWITCH followed by an Integer from 0 to 26.
2. *Alphabet-Name-1*, *Alphabet-Name-2*, *Class-Name-1*, *Class-Name-2*, *Condition-Name-1*, *Condition-Name-2*, *Event-Status*, *Mnemonic-Name*, *Object-Reference*, *Screen-Control*, *Symbolic-Character* and *SystemName* are User-defined words, as defined in the Definitions section in the Preface of this document.
3. *Literal-1*, *Literal-2*, *Literal-3*, *Literal-4*, *Literal-5*, *Literal-6*, *Literal-7* and *Literal-8* are Literals, as defined in the Definitions section in the Preface of this document.
4. *Crt-Status* is an unsigned integer Numeric Data Itemor a group Data Item.
5. *Class-Name-2* is a Data Item that must refer to a class for object handling.
6. *Object-Reference* is a Data Item that must refer to a class.
7. *Event-Status* is a Data Item whose structure must match the structure of the data item event-status defined in iscrt.def.
8. *Screen-Control* is a Data Item whose structure must match the structure of the data item screen-control defined in iscrt.def.

General Rules

1. *Mnemonic-Name* can be used in the SET Statement to alter the status of *Switch-Name*.
2. *Condition-Name-1* and *Condition-Name-2* can be used in a conditional expression to test the status of *Switch-Name*. *Condition-Name-1* is true when *Switch-Name* status is ON, *Condition-Name-2* it is OFF.
3. When the ALPHABET Phrase is specified, a reference to a collating sequence is defined.
    a. ASCII, STANDARD-1 and STANDARD-2 specify the ASCII character set.
    b. EBCDIC and NATIVE specify the EBCDIC character set.
4. if *Literal-1* is specified, *Alphabet-Name-1* will refer to a collating sequence.
    a. 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.
    b. When Literals here specified are Integers, they represent the ordinal number of a character within the native character set
    c. When they are Nonnumeric Literals, each character in the literal, starting with the leftmost character, is assigned successive ascending positions in the collating sequence being specified
    d. Any characters within the native collating sequence, which are not explicitly specified in the literal phrase, assume a position, in the collating sequence being specified, greater than any of the explicitly specified characters. The relative order within the set of these unspecified characters is unchanged from the native collating sequence.
    e. 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.

f. 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. 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.

5. 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`.
6. 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`.
7. When specified as literals in the `SPECIAL-NAMES` paragraph, the figurative constants `HIGH-VALUE` and `LOW-VALUE` are associated with those characters having the highest and lowest positions, respectively, in the native collating sequence.
8. When the `CLASS` phrase is specified, *Class-Name-1* consists if the exclusive set of characters defined by *Literal-4* and, optionally, *Literal-5*.

   a. If *Literal-4* and *Literal-5* are Integers, they represent the ordinal number of a character within the native character set.

   b. If the `THROUGH` phrase is specified, the contiguous characters in the native character set beginning with the character specified by the value of *Literal-4*, and ending with the character specified by the value of *Literal-5*, are included in the set of characters identified by *Class-Name-1*. In addition, the contiguous characters specified by a given `THROUGH` phrase may specify characters of the native character set in either ascending or descending sequence.
9. When the `CONSOLE` phrase is specified, all ANSI-style DISPLAY Statements will send information to the screen instead of sending them to the default output channel.
10. When the `CRT STATUS` phrase is specified, the value of the data item identified by *Crt-Status* will be updated with the last Exception or Termination value. If *Crt-Status* is defined as a group Data Item, it must have the following structure.

```
01 CRT-STATUS.
   05 CRT-KEY-1   PIC X.
   05 CRT-KEY-2   PIC X COMP-X.
   05 CRT-KEY-3   PIC X COMP-X.
```

The first two fields are set as follows:

CRT-KEY-1	CRT-KEY-2	Meaning
"0"	"0"	Termination key pressed
"0"	"1"	Auto-skip out of last field
"1"	x"00" - x"FF"	Exception key pressed
"3"	x"00"	Statement timed out
"9"	x"00"	No items fall within screen

If CRT-KEY-1 is "1", then CRT-KEY-2 contains the exception key value of the key that was pressed.

The third character always contains the same value that is returned by the CONTROL KEY phrase of the ACCEPT statement, if this value is in the range of 0 to 255. This is the same value returned when *Crt-Status* refers to a numeric data item instead of a group item.
When CRT-KEY-1 is set to "0", a normal termination has occurred. Any other value indicates an exception condition.

11. When the `CURRENCY` phrase is specified, *Literal-7* represents the currency symbol to be used in the PICTURE clause, in place of the default "$". The literal must be nonnumeric and is limited to a single character. It may be any character from the native character set except one of the following:

    a. digits 0 through 9;

    b. alphabetic characters consisting of the uppercase letters A, B, C, D, P, R, S, V, X, Z; the lowercase letters a through z; or the space;

    c. special characters * + - , . ; ( ) " = / '

    If *Literal-7* is of class national, the associated currency symbol may be used only to define a numeric-edited item with usage national. However in some cases non-national currency symbol can be correctly displayed even when used with plain usage display items, whether the OS uses an 8-bit codepage and the currency character is included in it.

12. When the `DECIMAL-POINT` phrase is specified, the functions of comma and period are exchanged in the character-string of the PICTURE clause and in numeric literals.

13. When the Cursor phrase is specified, the numeric data-item specified after this clause contains the cursor position on the screen. It works on both character-based and graphical screens. *cursor-name* can be a four or six character data-term. When it is 4 characters long, then the first two digits represent the line number, while the last two digits the column number. When it is 6 characters long, then the first three digits represent the line number, while the last three digits the column number.

14. When the `EVENT OBJECT` phrase is specified, *Class-Name-2* identifies the class that will handle events. Currently, only references to "java.util.EventObject" are accepted.

15. When the `EVENT SOURCE` phrase is specified, *Object-Reference* identifies the object that fired the event. Currently, only references to "com.iscobol.gui.server.CobolGUIJavaBean" are accepted.

16. When the `EVENT STATUS` phrase is specified, the data item identified by *event-status* will be used for event handling. Refer to the Event handling section of the isCOBOL Evolve: GUI Reference Guide for further details.

17. When the `SCREEN CONTROL` phrase is specified, the data item identified by *screen-control* will be used for handling the ACCEPT Statements. Refer to the Embedded Procedures section of the isCOBOL Evolve: GUI Reference Guide for further details.

18. When the `SYMBOLIC` phrase is specified, *Symbolic-Character* defines a figurative constant representing *Literal-8*.

19. If *Literal-8* is an Integer, it represents the ordinal number of a character within the native character set.

20. When the `IN` phrase is specified, and *Literal-8* is an Integer, it represent the ordinal number of a character within the character set identified by *Alphabet-Name-2*.

21. When the SIGN IS TRAILING SEPARATE phrase is specified, all signed data-items in the program Data Division store the sign in a separate byte.

22. C01 - C12 are known as printer channels; they can be used for line and form-feeds (for exmaple form feed is writing to a new page). With the syntax C01 IS TO-TOP-OF-PAGE you just define TO-TOP-OF-PAGE as an alternate name for C01. In our code we would write WRITE FILE-REC FROM DATA-REC AFTER ADVANCING TO-TOP-OF-PAGE . The -cv option is required to compile this syntax. Printer channels are affected by the property iscobol.printer.channels.

SCREEN CONTROL

23. Before executing any Embedded Procedure, the following data items, defined in "iscrt.def", are updated:

```
01  screen-control is special-names screen control.
    03  accept-control                         pic 9.
    03  control-value                          pic 999.
    03  control-handle                         handle.
    03  control-id                             pic xx comp-x.
```

accept-control	Always set to zero.
control-value	The index of the current active control. That index is a progressive number that starts at 1 and is increased by one for each control that can be activated. Display-only controls (such as Labels) do not cause the number to increase.
control-handle	The handle of the current active control.
control-id	The ID of the current active control. The ID of the control is the value of the ID Property. If not explicitly set, a progressive number is assigned. Display-only controls (such as Labels) cause the number to increase.

After the last statement of an Embedded Procedure has been executed, the value of *accept-control* is evaluated and the program behaves as explained below. This means that the programmer can influence the program behavior changing the value of *accept-control*.

Value	Behavior
0	The program behavior is not changed, this is the default.
1	The control that has the same index set in *control-value* is activated. [A]
2	The ACCEPT Statement terminates and the termination value set in *control-value* is assigned to the special register CRT STATUS.
3	The ACCEPT Statement terminates and the exception value set in *control-value* is assigned to the special register CRT STATUS. This is the most common way to simulate that a Function key has been pressed.
4	The control that has the same ID set in *control-id* is activated. [B] If more controls have the same ID, one of them is activated arbitrarily. If *control-id* is invalid, then the focus is moved on the control whose ID is the nearest in a range of 1000 in both directions; if no control with a near ID is found in the range of 1000, then the focus is moved on the first control in the tab order.

[A][B]If the control is either invisible, disabled or protected, then the next enabled control in the screen tab order is activated. If the control doesn't exist, then the ACCEPT terminates with the crt status set to zero.

The value of *accept-control* is evaluated also before every ACCEPT of the screen. This means that the programmer can influence the program behavior changing the value of *accept-control*. The rules listed above are applied, with the following difference:

[A]If the control is invisible, disabled, protected or doesn't exist, then the first enabled control whose tab order is the nearest to *control-value* is activated.

[B]If the control is invisible, disabled, protected or doesn't exist, then the first enabled control whose ID is the nearest to *control-id* is activated. If there are two controls whose ID value has the same distance to the requested ID, then the control with higher ID is activated.

Example

```
       WORKING-STORAGE SECTION.
       copy "iscrt.def".
       77  key-status special-names crt status pic 9(5).

       SCREEN SECTION.
       01  Screen-1.
           03 Ef-1 entry-field line 2, col 2, id 101 after MOVE-CURSOR.
           03 Ef-2 entry-field line 4, col 2, id 102.
           03 Ef-3 entry-field line 6, col 2, id 103 enabled 0.
           03 Ef-4 entry-field line 8, col 2, id 501.

       PROCEDURE DIVISION.

       MAIN.
           display Screen-1.

           accept  Screen-1 until key-status = 27
                on exception
                   if key-status = 5
                      perform MOVE-CURSOR
                   end-if
           end-accept.

       MOVE-CURSOR.
           move 103 to control-id.
           move 4   to accept-control.
```

Running the above program, when the cursor is on Ef-1,

- if the user presses Tab, the focus goes on Ef-4,
- if the user presses F5, the focus goes on Ef-2.

EVENT STATUS

24. Before executing an Event Procedure, the following data items, defined in iscrt.def are updated:

```
01  event-status is special-names event status.
    03  event-type                             pic x(4) comp-x.
    03  event-window-handle                    handle of window.
    03  event-control-handle                   handle.
    03  event-control-id                       pic xx comp-x.
    03  event-data-1                           signed-short.
    03  event-data-2                           signed-long.
    03  event-action                           pic x comp-x.
```

event-type	Event that has been fired. Almost all controls fire events. Please refer to the Events section of each control to see what events it fires and when. Symbolic names are defined in "isgui.def".
event-window-handle	Handle of the window that contains the control that fired the event.

event-control-handle	Handle of the control that fired the event. If the event has been fired by a window, it is set to zero.
event-control-id	ID of the control that fired the event. If the event has been fired by a window, it is set to zero.
event-data-1	Data item whose meaning may vary from event to event. Refer to the event documentation to see if and how to use *event-data-1*.
event-data-2	Data item whose meaning may vary from event to event. Refer to the event documentation to see if and how to use *event-data-2*.
event-action	Always set to event-action-normal.

After the last statement of an Event Procedure has been executed, the value of event-action is evaluated and the program behaves as explained below. This means that the programmer can influence the program's behavior by changing the value of event-action. Symbolic names listed below are defined in "isgui.def".

event-action-normal	This is the default value. The ACCEPT terminates if the event is a termination event.
event-action-terminate	This value causes the ACCEPT to terminate.
event-action-continue	This value causes the ACCEPT to continue even if the event is a termination event.
event-action-ignore	The event is ignored. At the end of the Event Procedure, the program returns directly to the ACCEPT statement and no further processing is done. The use of event-action-ignore is discouraged, unless stated otherwise.
event-action-fail	The event fails. The program will behave as if the event never occurred. The use of event-action-fail is discouraged, unless stated otherwise.
event-action-complete	By setting this value, the programmer communicates to the Framework that the handling of the event is complete and no further processing is needed. The use of event-action-complete is discouraged, unless stated otherwise.
event-action-fail-terminate	This value combines the behaviors of event-action-fail and event-action-terminate. The use of event-action-fail-terminate is discouraged, unless stated otherwise.

See Controls Reference for additional information about the EVENT-ACTION behavior in each control event.

REPOSITORY Paragraph

The `REPOSITORY` Paragraph contains class references.

General format

```
REPOSITORY.

  [ {CLASS     } Class-Name-1 AS Literal-1 ] ...
    {INTERFACE}
```

Syntax rules

1. *Class-Name-1* is a User-defined word, as defined in the Definitions section in the Preface of this document.
2. *Literal-1* is a Literal, as defined in the Definitions section in the Preface of this document.

General Rules

1. *Class-name-1* is the name of a class that may be used throughout the scope of the containing environment division.
2. *Literal-1* is the externalized name by which the class is known to the operating environment.

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. It should not be specified in the `CLASS-ID` Paragraph.

General format

```
INPUT-OUTPUT SECTION.

  [ FILE-CONTROL Paragraph ]

  [ I-O-CONTROL Paragraph ]
```

FILE-CONTROL Paragraph

The `FILE-CONTROL` paragraph allows specification of file-related information.

General format

```
[ FILE-CONTROL. ]

  [ { Indexed File    } ] ...
    { Relative File   }
    { Sequential File }
    { Sort File       }
    { XML File       }
```

Indexed File

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

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

Alternate record keys provide alternate means of retrieval for the records of a file. Such keys are named in the ALTERNATE RECORD KEY clause of the FILE-CONTROL Paragraph. 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.

```
SELECT [ {OPTIONAL    } ] File-Name
       [ {NOT OPTIONAL} ]

  ASSIGN TO [Device-1] [File] ORGANIZATION IS INDEXED

  [ WITH {COMPRESSION} ... ]
         {ENCRYPTION }
 [ COMPRESSION CONTROL VALUE IS Compression-Factor ]

  [ ACCESS MODE IS {SEQUENTIAL} ]
                   {RANDOM    }
                   {DYNAMIC   }

  [ COLLATING SEQUENCE IS Alphabet-Name ]

  [ FILE STATUS IS File-Status ]

  [ LOCK MODE IS { EXCLUSIVE WITH [MASS-
UPDATE]                                } ]
                 { {AUTOMATIC} [ WITH LOCK ON [MULTIPLE] {RECORD } ] [ [WITH] ROLLBACK
 ] }
                   {MANUAL   }                           {RECORDS}

  RECORD KEY IS { Data-Item-1                        } [ WITH [NO] DUPLICATES ]
                { Key-Name-1 [ = {Data-Item-2} ... ] }

  [ ALTERNATE RECORD KEY IS { Data-Item-
3                         } [ WITH [NO] DUPLICATES ] ] ...
                             { Key-Name-2 [ = {Data-Item-4} ... ] }

  [ RESERVE {Integer} [{AREA }] ]
                      {AREAS}

  [ CLASS IS Class-Name ] .
```

Relative File

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 comprised of a serial string of areas, each capable of holding a logical record. Each of these areas is denoted 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.

```
SELECT [ {OPTIONAL    } ] File-Name
       [ {NOT OPTIONAL} ]

  ASSIGN TO [Device-1] [File] ORGANIZATION IS RELATIVE


  [ ACCESS MODE IS { SEQUENTIAL [ RELATIVE KEY IS Relative-Key ] } ]
                   { RANDOM RELATIVE KEY IS Relative-Key         }
                   { DYNAMIC RELATIVE KEY IS Relative-Key        }

  [ FILE STATUS IS File-Status ]
  [ LOCK MODE IS { EXCLUSIVE                                          } ]
                 { {AUTOMATIC} [ WITH LOCK ON [MULTIPLE] {RECORD } ] }
                   {MANUAL   }                           {RECORDS}

  [ RESERVE {Integer} [AREA ] ]
                      [AREAS]
  [ WITH { COMPRESSION } ... ]
         { ENCRYPTION  }
 [ COMPRESSION CONTROL VALUE IS Compression-Factor ]

  [ CLASS IS Class-Name ] .
```

Sequential File

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

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

Variable-length binary sequential records occupy only as much disk space as necessary. If the maximum record size is equal to or less than 65,535 bytes, two bytes indicating record size are placed in front of each record when it is written to disk. If the record size is larger than 65,535 bytes, four bytes are placed in front of each record. This two- or four-byte field is not specified in your COBOL program, and non-COBOL programs that access the records need to be aware of the extra bytes.

In sequential files whose organization is `LINE SEQUENTIAL`, each record is terminated by one or more characters. They are not part of the data record, they are automatically appended to the record after writing and removed after reading.

Sequential files can be created either on disk or in memory.

```
SELECT [ {OPTIONAL    } ] File-Name
       [ {NOT OPTIONAL} ]

  ASSIGN TO [Device-2] [File] [ ORGANIZATION IS [ {BINARY} ] SEQUENTIAL ]
                                                {LINE   }
                                                {RECORD}

  [ ACCESS MODE IS SEQUENTIAL ]

  [ FILE STATUS IS File-Status ]

  [ LOCK MODE IS {EXCLUSIVE} WITH LOCK ON [MULTIPLE] {RECORD}
                                                     {RECORDS} ]
                 {AUTOMATIC}
                 {MANUAL   }

  [ PADDING CHARACTER IS Padding-Character ]
  [ WITH { COMPRESSION } ... ]
         { ENCRYPTION  }
 [ COMPRESSION CONTROL VALUE IS Compression-Factor ]

  [ { RECORD  } DELIMITER IS STANDARD-1 ]
    { RECORDS }

  [ CLASS IS Class-Name ].
```

Sort File

A sort file is a collection of records to be sorted by a SORT Statement. The sort file has no label procedures which the programmer can control and the rules for blocking and for allocation of internal storage are peculiar to the SORT Statement. The RELEASE and RETURN Statements imply nothing with respect to buffer areas, blocks, or reels. A sort file, then, may be considered as an internal file which is created (RELEASE Statement) from the input file or procedure, processed (SORT Statement), and then made available (RETURN Statement) to the output file or procedure.

```
SELECT [ {OPTIONAL    } ] File-Name
       [ {NOT OPTIONAL} ]

  ASSIGN TO [Device-3] [File]

  [ ACCESS MODE IS SEQUENTIAL ]

  [ { FILE } STATUS IS File-Status ]
    { SORT }

  [ LOCK MODE IS {EXCLUSIVE} WITH LOCK ON [MULTIPLE] {RECORD}
                                                     {RECORDS} ]
                 {AUTOMATIC}
                 {MANUAL   }

  [ PADDING CHARACTER IS Padding-Character ]
  [ CLASS IS Class-Name ]
  [ COLLATING SEQUENCE IS Alphabet-Name ]
```

XML File

XML file is an XML (Extensible Markup Language) file data file that is created by World Wide Web Consortium (W3C). XML is a language that can be read and utilized by computer software. It includes a formatted dataset that is planned to be processed by a website, web application, or software program.

XML files are plain text files, which means that they don't do anything in and of themselves except the transportation, structure, and storage of data. Unlike HTML, XML allows developers to structure data by using custom tags.

The flexibility of XML files make it an ideal option for cataloging information about almost all related items.

```
SELECT [ {OPTIONAL     } ] File-Name
       [ {NOT OPTIONAL} ]

  ASSIGN TO [ [ NOT ] LINE ADVANCING ] [File]  ORGANIZATION IS XML

  [ DOCUMENT-TYPE IS { EXTERNAL Doc-Type }
                     { OMITTED          }

  [ CHECK VALIDITY ON { INPUT  }
                      { OUTPUT }

  [ FILE STATUS IS File-Status ].
```

Syntax rules

1. *Alphabet-Name*, *Device-1*, *Device-2*, *Device-3*, *File-Name*, *Key-Name-1* and *Key-Name-2* are User-defined words, as defined in the Definitions section in the Preface of this document.
2. *File*, *Class-Name* and *Doc-Type* can be either Data Items or Nonnumeric Literals, as defined in the Definitions section in the Preface of this document.
3. *File* may contain quotes.
4. *Compression-Factor* and *Integer* are Integers, as defined in the Definitions section in the Preface of this document. The value must range from 1 and 100.
5. *Data-Item-1*, *Data-Item-2*, *Data-Item-3* and *Data-Item-4* are Data Items that must appear in the file description entry of the file that they belong to.
6. *Data-Item-5*, *File-Status* and *Relative-Key* are Data Items, as defined in the Definitions section in the Preface of this document.
7. *Padding-Character* is a one-character Nonnumeric Literal.
8. In Sort File, FILE STATUS and SORT STATUS are synonymous.

General Rules

1. The `OPTIONAL` phrase, if specified, indicates that the file need not be present when the program is run. If an optional file is opened in I-O or in EXTEND mode, it is created.
2. *Device-1* can be one of the following reserved words: `INPUT`, `OUTPUT`, `INPUT-OUTPUT`, `RANDOM`, `DISC`, `DISK`, `DYNAMIC`, `EXTERNAL`.
3. *Device-2* can be one of the following reserved words: `INPUT`, `OUTPUT`, `INPUT-OUTPUT`, `RANDOM`, `DISC`, `DISK`, `DYNAMIC`, `EXTERNAL`, `LINE ADVANCING`, `PRINT`, `PRINTER`, `PRINTER-1`, `ADDRESS`.
   `LINE ADVANCING`, `PRINT`, `PRINTER` and `PRINTER-1` are synonymous and must be specified for print files.
4. *Device-3* can be one of the following reserved words: `SORT`, `MERGE`, `SORT-MERGE`, `SORT-WORK`. These words are treated as commentary.

5. When INPUT, OUTPUT or INPUT-OUTPUT is specified as device phrase for a line sequential file, trailing spaces are removed from records before they are added to the file.
6. If *Device-2* is not specified, then DISC is assumed, unless -flsu compiler flag is used.
7. *File* is the name of the file to be opened, changing its value when the file has already been opened has no effect. The runtime Framework follows the rules described in Conceptual Characteristics of a File to resolve the name of the file.
8. When DYNAMIC is specified, if *File* refers to a variable that is not otherwise defined, the Compiler creates a Working-Storage variable by that name that is PIC X(256). It is the program's responsibility to move a valid file name to this data item prior to opening the file.
9. When EXTERNAL is specified, the runtime searches for *File* in the locations specified in environment variables and configuration properties. If a variable whose name is *File* is found, then its value is used for the physical file name otherwise *File* is used as is. There is a small difference between the EXTERNAL clause and the iscobol.file.env_naming (boolean) configuration property. When using EXTERNAL, the search of *File* in the configuration is performed by the file handler, while when using iscobol.file.env_naming (boolean), the search of *File* in the configuration is performed by the Framework before passing the file name to the file handler.
   If -cv compiler option is used, this name is processed first by ignoring any characters that appear before the last hyphen in the word (including the hyphen itself). For example:

```
ASSIGN TO EXTERNAL EX-1-MYFILE
```

   results in "MYFILE" being used for the file name.
10. When ADDRESS is specified, the runtime searches for *File* in the Data Division items. The content of the sequential file is stored in the data item and resides in memory. The data item pointed to by *File* should be alphanumeric. If you wish to share the memory file between two programs in the same run unit, then *File* should be defined as EXTERNAL or passed as parameter through the Linkage Section.
11. Each *File-Name* in the Data Division must be specified only once in the FILE-CONTROL paragraph. Each *File-Name* specified in the SELECT clause must have a file description or a sort description entry in the Data Division of the same program.
12. When the ORGANIZATION Phrase is not specified, the compiler assumes that the file is sequential.
13. COMPRESSION activates file compression. Note that this feature is not supported by every indexed file handler. For example: c-tree and VisionJ supports it, while JIsam does not support it. Refer to the documentation of the chosen file handler to know if file compression is supported or not.
14. ENCRYPTION activates file encryption. Note that this feature is not supported by every indexed file handler. For example: c-tree and JIsam supports it, while VisionJ does not support it. Refer to the documentation of the chosen file handler to know if file encryption is supported or not.
15. COMPRESSION and ENCRYPTION can't be used together on the same file.
    Note - c-tree allows you to activate file compression and encryption also via configuration. See Configuring the client for details. Activating file compression and encryption via configuration has a couple of advantages: you can activate both compression and encryption on the same file and you can customize algorithms, strategies and levels.
16. When the ACCESS Phrase is not specified, the compiler assumes that access mode is sequential.
17. When the SEQUENTIAL Phrase is specified, the file is accessed in a sequential order. The START Statement may be used to establish a starting point for a series of subsequent sequential retrievals.
18. When the RANDOM Phrase is specified, the file is accessed in a programmer-specified order. The I-O Statements may be used to retrieve or store a specific record in the file. Sequential access is not permitted.
19. When the DYNAMIC Phrase is specified, the file is accessed either in a sequential or in a programmer-specified order, depending on the I-O statement being executed.

20. When the COLLATING Phrase is specified, the file will use the collating sequence specified by *Alphabet-Name*. The behavior depends on the file system being used.
21. *File-Status* is a two-character conceptual entity whose value is set to indicate the status of an input-output operation during the execution of an I-O statement and prior to the execution of any imperative statement associated with that I-O statement or prior to the execution of any applicable USE AFTER STANDARD EXCEPTION procedure.
22. When the LOCK Phrase is not specified, the lock mode is automatic.
23. When the EXCLUSIVE Phrase is specified, the lock mode is exclusive. The entire file is locked and no other processes can have access to it
24. When the MASS-UPDATE Phrase is specified, the file access is optimized for heavy updates. The actual behavior depends on the file system being used.
25. When the AUTOMATIC Phrase is specified, the lock mode is automatic. Records are locked when any READ Statement is executed.
26. When 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.
27. When the MULTIPLE Phrase is specified in the LOCK ON Phrase, simultaneous locks on multiple records of the same logical file are allowed.
28. The setting of a record lock is part of the atomic operation of an I-O statement.
29. When the ROLLBACK clause is specified, WITH LOCK ON MULTIPLE RECORDS will automatically be in effect. However, if you compile with the -fl option, you must specify multiple locking rules for the files that need them.
30. When the ROLLBACK clause is specified, the runtime will automatically effect a START TRANSACTION before opening the file, and a COMMIT after opening it. Thus, every OPEN of the file will automatically be done within a transaction.
31. *Data-Item-1* and *Key-Name-1* identify the prime record key of the file.
32. *Data-Item-3* and *Key-Name-2* identify one of the alternate keys of the file.
33. When the WITH DUPLICATES Phrase is specified, duplicate values are allowed for the key.
34. When the WITH NO DUPLICATES Phrase is specified, only unique values are allowed for the key.
35. When the WITH DUPLICATES Phrase is specified in the RECORD KEY Phrase, the actual behavior depends on the file system being used.
36. The RESERVE clause is treated as a commentary.
37. *Class-Name* identifies the class that handles the files. It is the file system handler.
38. The PADDING clause is treated as a commentary.
39. LOCK MODE is ignored for Sequential and Sort files.
40. Class-Name specifies the internal Framework class that will handle the file. There is a class for each known file.index. Setting the CLASS clause to a specific value forces the file to be handled by a specific internal class, overriding existing settings like Framework properties and Compiler options. The CLASS clause must point to a valid class, the class must exists and be suitable for the file type, otherwise an error will be returned. Possible values are listed in the Class-Name column of the File Handlers tables.
    If the CLASS clause is omitted, indexed files are handled based on the iscobol.file.index configuration property, binary files are handled based on the iscobol.file.sequential configuration property, relative files are handled based on the iscobol.file.relative configuration property and line sequential files are handled based on the iscobol.file.linesequential configuration property.
41. isCOBOL sorting logic for SORT files is totally written in Java and is configurable. The maximum size of data to sort is equal to the maximum size of a Java file (up to 2^64 bytes depending on O.S.). It uses a predefined block of RAM memory that is 1 MB in size by default, but can be configured by setting the iscobol.sort.memsize configuration property.

If this size is enough, the data is entirely sorted using the Collections Java object.
If the size of the memory block is not enough, the block is saved on disk in a temporary directory (the system temporary directory by default, but it can be configured by setting the iscobol.sort.dir configuration property), then the memory block is cleared and used for remaining data.
There is a maximum number of temporary files (16 by default, but it can be configured by setting the iscobol.sort.maxfiles configuration property). If this number is not enough, then all the sorted files are merged into a single sorted temporary so the Framework can create more temporary files.
The process continues until no more data to sort is available. At the end, if the data size is less than the allocated RAM memory, the results are returned directly from memory, otherwise data is read from temporary files.
With the default configuration, optimal performances are obtained when the data to sort is less than 1 MB.

42. When DOCUMENT-TYPE is not specified or is OMITTED, no schema is associated with the XML file.
43. The CHECK clause can be repeated multiple times. For example:

```
           SELECT XML-FILE ASSIGN TO "file.xml"
                  ORGANIZATION XML
                  CHECK INPUT
                  CHECK OUTPUT
                  FILE STATUS XML-FS.
```

File Names

File Names represents the name of the file to be opened. The rules for resolving the name of the file are described below.

1. If *File* starts with "-F", all file operations are executed on the physical file indicated in *File*. -F causes the file name to be used as is, without applying iscobol.file.prefix and iscobol.file.suffix to it.
2. If *File* starts with "-E " or if iscobol.file_env_naming=1, then *File* contains the name of a configuration property that defines the physical name of the file.
3. If *File* starts with "isf://*server-name*:*serverport*: " or just "isf://*server-name*: ", the next characters identify the name of a file that is managed by the File Server. See The ISF protocol for more details.
4. If *File* is "-P SPOOLER", all file operations are redirected to the print spooler.[A]
5. If *File* is "-P PREVIEW", all file operations are redirected to the preview manager that shows the print preview. The preview window allows job to be saved into a pdf file on disk.[A]
6. If *File* is "-P PDF filename", all file operations are redirected to the PDF manager that generates the PDF file named filename; the default name (when filename is not indicated) is "iscoboljob.pdf". [A]
7. If *File* is "-P SPOOLER-DIRECT", all file operations are directly redirected to the printer.[A]
8. If *File* is "-P ###", where ### doesn't match with any of the above, ### is passed to the current system command interpreter and the file allows the command output stream and input stream to manage, depending on the open mode.[A]
9. If *File* is "-S IN", all file operations are redirected to the standard-input device.
10. If *File* is "-S OUT", all file operations are redirected to the standard-output device.
11. If *File* is "-S ERR", all file operations are redirected to the standard-error device.
12. If *File* is "-Q <printerName>", all file operations are redirected to the named printer. Details are explained in -Q <printerName> syntax.
13. If *File* is "PRINTER?", all file operations are redirected to the print spooler in RM/COBOL compatibility mode. It differs from "-P SPOOLER" in terms of text positioning rules.

Data is converted using the current encoding while working with print files. You can avoid this conversion by using + instead of - in file assignments (i.e. +F, +P SPOOLER, +S OUT, etc.)

[A] Piping is allowed only for

- print files; files that are implicitly or explicitly ASSIGN TO PRINTER
- line sequential files; only if the program is compiled with -flsu option

-Q <printerName> syntax

The "-Q <printerName>" syntax allows to assign a print file to a specific printer and configure some settings like font and orientation through the physical file name.

The base syntax assigns the file to a named printer:

```
-Q <printerName>
```

For example, the below print file will work on a printer named "Samsung-ML-3471ND":

```
SELECT print-file ASSIGN TO PRINT "-Q Samsung-ML-3471ND"
       ORGANIZATION LINE SEQUENTIAL
```

The extended syntax allows to set one or more additional settings as follows:

```
-Q
   <printerName>;DIRECT=<value>;FONT=<value>;PITCH=<value>;LINES=<value>;ORIENTATION=
   <value>;COPY=<value>
```

Settings have the following meaning:

DIRECT	Two possible values are allowed: “ON”: all file operations are directly redirected to the printer. “OFF”: all file operations are redirected to the print spooler.
FONT	Specifies the name of the font. Spaces are allowed so double quotes must not be used in order to delimit the font name.
PITCH	Specifies the size of the font
LINES	Specifies the number of lines for each page. When this number is reached, a new page is printed.
ORIENTATION	Three possible values are allowed: 0: default printer orientation 1: portrait orientation 2: landscape orientation
COPY	Specifies the number of copies.

Any setting in addition to the printer name is optional and can appear in any order.

The following complex string, for example, assigns a print file to the printer named "Samsung-ML-3471ND" and specifies that the print job will have a landscape orientation and use the Courier New font:

```
-Q Samsung-ML-3471ND;FONT=Courier New;PITCH=10;ORIENTATION=2
```

File names interpretation

The isCOBOL Framework constructs the full-name of a file in this way:

1. If iscobol.file.env_naming (boolean) is set in the configuration, the physical file name is searched between the environment variables. During this search hyphens are translated to underscores and names are made lowercase to be case insensitive before the comparison between file name and configuration property name.
   Note: the conversion affects only properties set through SET ENVIRONMENT. For properties set in the external configuration, it's your responsibility to use underscores and lowercase names, otherwise they will not match.
2. The physical name is made upper-case or lower-or depending on iscobol.file.case setting.
3. If iscobol.file.remove_name_spaces (boolean) is set to true, then spaces are removed from the file name.
4. If iscobol.file.index.strip_extension (boolean) is set to true and the file name includes an extension, then the extension is removed (only for indexed files).
5. Unless the file name starts with either "-F" or "+F", the extension specified by iscobol.file.suffix property is appended to the resulting name.
6. Unless the file name starts with either "-F" or "+F", the resulting name is appended to the iscobol.file.prefix paths.
7. If the resulting name is not an absolute path, it is appended to the working directory specified by the C$CHDIR routine, if any.

Once the full-name is ready, it's passed to the file handler.

Note - if the file is indexed and the iscobol.file.index.open_hook * configuration property is set the full-name is passed to a hook program, which might alter the file-name.

Some file handlers might perform additional operations on the file name. For example:

Jlsam

- Jlsam automatically appends the .dat extension to the data file name and the .idx extension to the index file name before opening the disc file. These extensions can be configured by iscobol.file.index.data_suffix * and iscobol.file.index.index_suffix * properties.
- If the full-name constructed by isCOBOL is a relative path, the path will be relative to the current working directory.

c-tree

- c-tree automatically appends the .dat extension to the data file name and the .idx extension to the index file name before opening the disc file. These extensions can be configured by <datafilesuffix> and <indexfilesuffix> settings in the c-tree configuration.
- If the full-name constructed by isCOBOL is a relative path, the path will be relative to the server working directory (see SERVER_DIRECTORY and LOCAL_DIRECTORY for details).

Database Bridge

- Database Bridge converts all hypens and dots to underscore in the full-name before accessing the table.
- paths are not considered unless iscobol.easydb.dirlevel property is set to a value greater than zero.

remote

- if the File Server machine is a different platform than the local PC (e.g. Linux server and Windows client), a full path built on the local PC may not be understood correctly by the server (e.g. Linux can't interpret "C:" at the beginning of a path), therefore you should set the FILE-PREFIX according to the server platform as the full path will be resolved server side.

I-O-CONTROL Paragraph

The FILE-CONTROL paragraph allows specification of input-output rules for the files described in the program.

General format

```
I-O-CONTROL.

  [ APPLY { LOCK-HOLDING } ON {File-1} ... ] ...
          { WRITE-ONLY   }

  [ SAME {RECORD } AREA FOR {File-2} ... ] ... .
         {RECORDS}
```

Syntax rules

1. *File-1* and *File-2* must be specified in the FILE-CONTROL Paragraph.

General Rules

1. When the APPLY LOCK-HOLDING phrase is specified, multiple record locking rules are applied to *File-1*.
2. The APPLY WRITE-ONLY and SAME clauses are treated as a commentary.

Chapter 4

Data Division

The Data Division describes the data that is to be processed by the object program.

General format

```
DATA DIVISION.

  [ FILE Section ]

  [ WORKING-STORAGE Section ]

  [ THREAD-LOCAL-STORAGE Section ]

  [ LOCAL-STORAGE Section ]

  [ LINKAGE Section ]

  [ REPORT Section ]

  [ SCREEN Section ]
```

FILE Section

The File Section defines the structure of data files. Each file is defined by a file description entry and one or more record description entries. Record description entries are written immediately following the file description entry.

General format

```
FILE SECTION.


  [ File-Description {Record-Description} ... ] ...
```

File description

In a COBOL program the file description entry (FD or SD 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/SD), a file-name, and a series of independent clauses. The clauses of a file description entry specify the size of the logical and physical records, the names of the records which comprise the file, and finally the number of lines to be written on a logical printer page.

General format

```
Level-Indicator File-Name
   [ IS EXTERNAL [ AS External-Name ] ]

   [ IS GLOBAL ]

   [ IS THREAD-LOCAL ]

   [ CODE-SET  [IS ] Alphabet-1 ]

   [ LABEL {RECORD } [IS ] {STANDARD} ]
           {RECORDS} [ARE] {OMITTED }

   [ 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 Depend ] }
            { CONTAINS Integer-6 TO Integer-
7 CHARACTERS                                          }

   [ VALUE OF LABEL IS Literal-1 ]

   [ VALUE OF {FILE-ID} IS Id-Name ]
              {ID     }

   [ VALUE OF Literal-1 IS {Literal-2  } ]
                          {Data-item-1}

   [ LINAGE IS Page-Lines LINES

     [WITH FOOTING AT Footing-Line ]

     [LINES AT TOP Top-Lines ]

     [LINES AT BOTTOM Bottom-Lines ] ]

   [ DATA {RECORD  IS } {Record-Name} ... ]
          {RECORDS ARE}
   [ RECORDING MODE IS {V}          ]
                       {F}
                       {U}
                       {S}
                       {VARIABLE}
                       {FIXED}
   [ {REPORT } [IS ] Report-Name ]
     {REPORTS} [ARE]
```

Syntax rules

1. *Level-Indicator* must precede *File-Name*.
2. *File-Name* is a User-defined word, as defined in the Definitions section in the Preface of this document.
3. *File-Name* must have a corresponding entry in the FILE-CONTROL Paragraph of the INPUT-OUTPUT Section in the Environment Division.

4. The clauses which follow *File-Name* may appear in any order.
5. *Integer-1*, *Integer-2*, *Integer-3*, *Integer-4*, *Integer-5*, *Integer-6* and *Integer-7* are Integers, as defined in the Definitions section in the Preface of this document.
6. *Depend* is a Numeric Data Item, as defined in the Definitions section in the Preface of this document.
7. *Id-Name* can be either a Data Item or a Nonnumeric Literal, as defined in the Definitions section in the Preface of this document.
8. *Page-Lines*, *Footing-Line*, *Top-Lines* and *Bottom-Lines* can be either Numeric Data Items or a Integers, as defined in the Definitions section in the Preface of this document.
9. V and VARIABLE are synonymous.
10. F and FIXED are synonymous.
11. Literal-1 and Literal-2 are non-numeric literals.
12. Data-Item-1 is an alphanumeric data item.
13. *Alphabet-1* is the name of an alphabet declared in the SPECIAL-NAMES Paragraph.
14. The CODE-SET clause may be associated only with a sequential file.
15. If the CODE-SET clause is used, then only USAGE DISPLAY items may appear in the file's record description entry. In addition, every signed numeric field must have a SIGN IS SEPARATE clause.
16. THREAD-LOCAL can't be specified along with EXTERNAL and GLOBAL.

General rules

1. *Level-Indicator* may be FD to identify the beginning of a file description entry, SD to identify the beginning of a sort file description entry or XD to identify the beginning of an XML file description entry.
2. When the `IS EXTERNAL` clause is specified, the file will be shared among all programs that declare it.
3. The `AS` phrase of the `EXTERNAL` clause is treated as a commentary.
4. The `IS GLOBAL` clause is treated as a commentary.
5. The `LABEL` clause is treated as a commentary.
6. `VALUE OF Literal-1` is treated as a commentary.
7. *Integer-1* must be less than or equal to *Integer-2*. The `BLOCK` clause is treated as a commentary.
8. The `RECORD` clause specifies the number of character positions in a fixed length record, or specifies the range of character positions in a variable length record. If the number of character positions does vary, the clause specifies the minimum and maximum number of character positions.
    a. If the `RECORD` clause is not specified, the size of each data record is completely defined in the record description entry. Relative Files cannot have variable record size, so the maximum record size is used.
    b. No record description entry for the file may specify a number of character positions greater than *Integer-3*.
    c. When *Integer-3* is specified, the record length is fixed. *Integer-3* specifies the number of character positions contained in each record in the file.
    d. Record descriptions for the file must not describe records which contain a lesser number of character positions than that specified by *Integer-4* nor records which contain a greater number of character positions than that specified by *Integer-5*.
    e. *Integer-4* must be less than or equal to *Integer-5*.
    f. The `VARYING` clause is used to specify variable length records. *Integer-4* specifies the minimum number of character positions to be contained in any record of the file. *Integer-5* specifies the maximum number of character positions in any record of the file.

g. The number of character positions associated with a record description is determined by the sum of the number of character positions in all elementary data items excluding redefinitions and renamings, plus any implicit FILLER due to synchronization. If a table is specified:

   i. The minimum number of table elements described in the record is used in the summation above to determine the minimum number of character positions associated with the record description.

   ii. The maximum number of table elements described in the record is used in the summation above to determine the maximum number of character positions associated with the record description.

h. If *Integer-4* is not specified, the minimum number of character positions to be contained in any record of the file is equal to the least number of character positions described for a record in that file.

i. If *Integer-5* is not specified, the maximum number of character positions to be contained in any record of the file is equal to the greatest number of character positions described for a record in that file.

j. If *Depend* is specified, the number of character positions in the record must be placed into the data item referenced by *Depend* before any RELEASE, REWRITE, or WRITE Statement is executed for the file.

k. If *Depend* 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 *Depend*.

l. During the execution of a RELEASE, REWRITE, or WRITE Statement, the number of character positions in the record is determined by the following conditions:

   i. If *Depend* is specified, by the content of the data item referenced by *Depend*.

   ii. If *Depend* is not specified and the record does not contain a variable occurrence data item, by the number of character positions in the record.

   iii. If *Depend* 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 I-O Statement.

m. If *Depend* is specified, after the successful execution of a READ or RETURN Statement for the file, the contents of the data item referenced by *Depend* will indicate the number of character positions in the record just read.

n. If the INTO Phrase is specified in the READ or RETURN Statement, the number of character positions in the current record that participate as the sending data items in the implicit MOVE Statement is determined by the following conditions:

   i. If *Depend* is specified, by the content of the data item referenced by *Depend*.

   ii. If *Depend* is not specified, by the value that would have been moved into the data item referenced by *Depend* had *Depend* been specified.

o. When *Integer-6* and *Integer-7* are specified, they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively. However, in this case, the size of each data record is completely defined in the record description entry.

p. The size of each data record is specified in terms of the number of character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of characters in all fixed length elementary items plus the sum of the maximum number of characters in any variable length item subordinate to the record. This sum may be different from the actual size of the record.

9. *Id-Name* is the name of the file to be opened. Its value overrides the value of *File* specified in the FILE-CONTROL Paragraph of the INPUT-OUTPUT Section in the Environment Division.

10. 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.

a. The LINAGE clause can be specified only for Sequential Files.

b. 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.

c. *Page-Lines* specifies the number of lines that can be written and/or spaced on the logical page. The value must be greater than zero. That part of the logical page in which these lines can be written and/or spaced is called the page body.

d. *Footing-Line* specifies the line number within the page body at which the footing area begins. The value must be greater than zero and not greater than *Page-Lines*.

   The footing area comprises the area of the page body between the line represented by *Footing-Line* and the line represented by *Page-Lines*, inclusive.

e. *Top-Lines* specifies the number of lines that comprise the top margin on the logical page. This area is not writable. The value may be zero.

f. *Bottom-Lines* specifies the number of lines that comprise the bottom margin on the logical page. This area is not writable. The value may be zero.

g. *Page-Lines*, *Top-Lines*, and *Bottom-Lines*, if specified, are used at the time the file is opened by the execution of an OPEN Statement with the OUTPUT Phrase, to specify the number of lines that comprise each of the indicated sections of a logical page. *Footing-Line*, 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.

h. The values of the data items referenced by *Page-Lines*, *Top-Lines*, and *Bottom-Lines*, if specified, are used as follows:

   i. The values of the data items at the time an OPEN Statement with the OUTPUT Phrase is executed for the file are used to specify the number of lines that are to comprise each of the indicated sections for the first logical page.

   ii. 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 to comprise each of the indicated sections for the next logical page.

i. *Footing-Line* is used to define the footing area for the first logical page at the time an OPEN Statement with the OUTPUT Phrase is executed for the file. 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.

j. 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:

   i. A separate LINAGE-COUNTER is supplied for each file described in the File Section whose file description entry contains a LINAGE clause.

   ii. LINAGE-COUNTER may be referenced only in Procedure Division statements; however only the input-output control system may change the value of LINAGE-COUNTER. Since more than one LINAGE-COUNTER may exist in a program, the user must qualify LINAGE-COUNTER by *File-Name* when necessary.

   iii. 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 increased incrementally to exceed the value specified by *Page-Lines*.
   - When the ADVANCING *Line-Count* Phrase of the WRITE Statement is specified, the LINAGE-COUNTER is increased incrementally by *Line-Count*.
   - When the ADVANCING Phrase of the WRITE Statement is not specified, the LINAGE-COUNTER is increased incrementally by the value one.
   - The value of LINAGE-COUNTER is automatically reset to one when the device is repositioned to the first line that can be written on for each of the succeeding logical pages.

   iv. 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.

k. Each logical page is contiguous to the next with no additional spacing provided.

11. The DATA RECORDS clause is treated as a commentary.
12. RECORDING MODE requires the -cv compiler option and is treated as a commentary.
13. *Report-Name* must be the same as the *Report-Name* used in the Report Description (RD) entry.
14. The CODE-SET clause associates a character set with a sequential file. If the character set is not the native character set, then a translation to the native set is also implied.
15. If the THREAD-LOCAL clause is specified, a separate copy of the file connector is created and set to its initial state for each new thread of execution that enters the program. The file connector is only visible to the thread causing its creation. The file connector is destroyed when the creating thread's execution terminates or when a CANCEL statement on the program is executed; otherwise, on subsequent calls to the program within that thread, the file connector is in its last used state.

Record Description

A record description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by the data-name or FILLER clause, if specified, followed by a series of independent clauses as required. A record description may have a hierarchical structure and therefore the clauses used within an entry may vary considerably, depending upon whether or not it is followed by subordinate entries.

General format

```
{Data-Description} ...
```

WORKING-STORAGE Section

The Working-Storage Section is located in the Data Division of a program, factory, object, or method.

Data described in the Working-Storage Section are static data. The Working-Storage Section describes records and subordinate data items that are not part of data files.

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

General format

```
{  [ {PUBLIC   } . ]  Data-Description }
     {PRIVATE  }
     {PROTECTED}
```

Syntax rules

1. The PUBLIC, PRIVATE and PROTECTED phrases may be used only in the OBJECT Paragraph.
2. When one of the optional phrases PUBLIC, PRIVATE or PROTECTED is specified, the subsequent data item must have level numbers 01 or 77.

General rules

1. When the PUBLIC phrase is specified, the subsequent data items and all their subordinate items can be used by any program using the object in which they are defined.
2. When the PRIVATE phrase is specified, the subsequent data items and all their subordinate items can be used only by the class containing the object in which they are defined.
3. When the PROTECTED phrase is specified, the subsequent data items and all their subordinate items can be used by the class containing the object in which they are defined and by those classes that inherit from it.
4. If neither PUBLIC nor PRIVATE nor PROTECTED are specified, data items are PRIVATE.
5. In the Data Division of a method, the data items are local or not depending if the LOCAL-STORAGE Section is present as well.
   If both the Working-Storage Section and the Local-Storage Section are present, then data items defined under the Local-Storage Section are local while data items defined under the Working-Storage Section are not local.
   If only the Working-Storage Section is present, then data items defined under the Working-Storage Section are local.

THREAD-LOCAL-STORAGE Section

The Thread-Local-Storage Section describes data which is unique to each thread, and is persistent across calls.

General format

```
{ Data-Description } ...
```

Syntax rules

1. The EXTERNAL and GLOBAL clauses in Data-Description are not permitted.

General rules

1. A separate copy of each data item in thread-local storage is created and set to its initial state for each new thread of execution that enters the program. The data item is only visible to the thread causing its execution. The data item is destroyed when the creating thread's execution terminates, or when a CANCEL statement on the program is executed; otherwise, on subsequent calls to the program within that thread, the data item is in its last used state.
2. In a single-threaded environment, thread-local storage behaves as working storage.

LOCAL-STORAGE Section

The Local-Storage Section is located in the Data Division of a program or method.

Data described in the Local-Storage Section are automatic data.

The Local-Storage Section is provided specifically for use in methods and recursive calls.

The Local-Storage Section is composed of the section header, followed by record description entries and/or data description entries for noncontiguous data items.

General format

```
{ Data-Description } ...
```

Syntax rules

1. The EXTERNAL and GLOBAL clauses in Data-Description are not permitted.

General rules

1. A separate copy of the Local-Storage Section is created each time the runtime element is activated and exists only during the lifetime of that activation.

LINKAGE Section

The Linkage Section appears in the called program and describes data items that are to be referred to by the calling program and the called program.

The Linkage Section in a program is meaningful if and only if the object program is to function under the control of a CALL Statement or CHAIN Statement, and the statement in the calling program contains a `USING` Phrase.

The Linkage Section is used for describing data that is available through the calling program but is to be referred to in both the calling and the called program. In the case of index-names, no such correspondence is established and index-names in the called and calling programs always refer to separate indices.

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

General format

```
{ Data-Description } ...
```

REPORT Section

The report section describes the reports to be written to report files. The description of each report begins with a report description (RD) entry and is followed by one or more report group descriptions.

General format

```
REPORT SECTION. [ report-description-entry {constant-entry                } ... ] ...
                                           {report-group-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 data items. Each of these items is defined in a separate data description entry that begins with the special level-number 77.

```
77-level data description entry
```

Syntax rules

1. The report section may be specified in a function definition or a program definition. Within a class definition, the report section may be specified only in a factory definition or an instance definition, but not in a method definition. The report section shall not be specified within an interface definition.

General format

```
RD report-name-1 [ IS GLOBAL ]
  [ CODE IS { identifier-1 } ]
            { literal-1    }

  [ { CONTROL IS   } { data-name-1 ... }        } ]
    { CONTROLS ARE } { FINAL [ data-name-1 ] ...}
  [  LINE LIMIT IS integer-1 ]
  [  PAGE { LIMIT IS   } integer-2 { LINE   } [ HEADING integer-3 ] ]
          { LIMITS ARE }           { LINES  }
  [  FIRST DETAIL integer-4 ]
  [  LAST DETAIL integer-5 ]
  [  FOOTING integer-6 ]
```

Syntax rules

1. There shall be one and only one REPORT clause specifying report-name-1 in a given file description entry.
2. The clauses that follow report-name-1 may appear in any order.

General rules

1. If GLOBAL is specified, report-name-1 and all its constituent report groups, its PAGE-COUNTER and LINE-COUNTER, and any sum counters defined in report-name-1 are global.
2. The CODE clause can be used to prefix non-printable fields to the report records.
3. The CONTROL clause should be coded in the RD if the report has additional lines, such as total lines and subheadings, that are to be produced upon a change of value in one or more "key" fields (known as control fields or simply controls).
4. The LINE LIMIT clause indicates the maximum number of columns likely to be required for the longest line of your report. It enables report writer to check that all of your report fields appear within the limits of the report line, and to warn you if there is any danger of data being lost beyond the right-hand extremity of the lines.
5. The PAGE LIMIT clause should be coded if the report is to be divided into pages. It also allows you to sub-divide the page into regions for the headings, main data, and footings.

Report description entry

The report description (RD) entry gives a name to the report and defines its physical and logical subdivisions. (See Physical subdivisions of a report.)

An RD entry shall be followed by one or more report group description entries. The RD entry and the report group description entries that follow fully describe one report. A report is in the active state after the successful execution of an INITIATE statement referencing that report description entry and before the successful execution of a TERMINATE statement referencing that report description entry. At any other time, it is in the inactive state.

Report group description entry

A report group is a block of zero, one, or more report lines that is treated, both logically and visually, as a single unit. Each report group description shall consist of a level 1 report description entry followed by zero, one, or more subordinate entries, describing the vertical and horizontal layout of the report group and the content or origin of each of its printed data items, known as printable items.

The report groups associated with the report are specified immediately following the report description entry. If several report groups are specified, the order in which they are defined is not significant. The first entry of each report group description has level number 1 and a TYPE clause and, if a data-name is also specified, this may be used subsequently to identify the report group. Further subordinate group and elementary entries may be specified to describe additional elements of the report group.

General Format

```
number [ entry-name-clause ]
       [TYPE clause]
       [NEXT GROUP clause]
       [LINE clause]
       [PICTURE clause]
       [ USAGE IS ] DISPLAY
                    NATIONAL ?
       [SIGN clause]
       [JUSTIFIED clause]
       [COLUMN clause]
       [BLANK WHEN ZERO clause]
       source-clause
       SUM clause
       VALUE clause
       PRESENT WHEN clause
       [GROUP INDICATE]
       [OCCURS [ integer-1 TO ] integer-2 TIMES
               [ DEPENDING ON data-name-1 ] [ STEP integer-3 ]]
       [VARYING clause]
```

Report subdivisions

A report has physical and logical subdivisions that interact to determine what is printed on a page.

Physical subdivisions of a report

Pages

Each report is composed of pages of equal size, defined by the PAGE clause, or one page of indefinite size. Each page heading, body group (detail, control heading, or control footing), and page footing appears in a separate subdivision of the page. Each report heading or report footing may appear in any position on the page. Advancing to a new page is executed automatically for body groups, including the printing of a page footing and page heading.

Lines

Each report group is divided vertically into zero, one, or more lines. Each line, or multiple line set, is represented in the report group description by a LINE clause. The NEXT GROUP clause, where defined, specifies additional vertical spacing following the report group.

Report Items

Each line of each report group is divided horizontally into zero, one, or more printable items. Each printable item, or adjacent set of printable items, is defined in the report group description by an elementary entry containing a COLUMN clause. The value that is placed in a printable item is determined solely by a SOURCE, SUM, or VALUE clause in its data description entry. In addition, unprintable items may be specified whose entries contain no COLUMN or LINE clause but from which printable items may be derived.

Report items shall not be accessed or referred to by any clauses in any other section of the data division or by any other procedure division statements, except in the following cases:

1. Sum counters may be inspected or altered in the procedure division.
2. Report groups of type DETAIL are accessed by the GENERATE statement.

A report item referenced in a function identifier or inline method invocation shall be an elementary report item.

Logical Subdivisions of a Report

Report groups of type DETAIL may be structured into a nested set of control groups. Each control group may begin with a control heading and end with a control footing.

A control break occurs when a change of value is detected in a control data item during the execution of a GENERATE statement. The hierarchy of control data items is used to check automatically for any such change in value. The detection of a control break causes the same GENERATE statement to print each defined control footing, in reverse hierarchical order, and each defined control heading, in hierarchical order.

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-Description } ...
```

Data Description

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

The initial value of data items is spaces, except for items possessing the external attribute whose initial value is binary zeros (NULL).

General format

Format 1

```
Level-Number {Data-Name-1 } { [REDEFINES clause]             }
             {FILLER      }   [IS EXTERNAL clause]
                              [IS EXTERNAL-FORM clause]
                              [IS IDENTIFIED clause]
                              [IS SPECIAL-NAMES clause]
                              [IS TYPEDEF clause]
                              [IS TYPEDEF clause]
                              PICTURE clause
                              {[USAGE clause]      }
                              {[GROUP-DYNAMIC clause]}
                              {[GROUP-USAGE clause]}
                              [SIGN clause]
                              [OCCURS clause]
                              [SYNCHRONIZED clause]
                              [JUSTIFIED clause]
                              [BLANK WHEN ZERO clause]
                              [VALUE clause]
                              [PROPERTY clause]

                            { SAME AS clause               }
```

Format 2

```
66 Data-Name-1 RENAMES clause
```

Format 3

```
88 Condition-Name-1

  {VALUE } [IS ] { Literal-2 [ {THROUGH} Literal-
3 ] } ... [ WHEN SET TO FALSE Literal-4 ] .
  {VALUES} [ARE]               {THRU   }
```

Format 4

```
78 Constant-Name-1

  [CONSTANT] VALUE IS { Nonnumeric-Literal-1                          } .
                      { Numeric-Literal-1 [ {+} Numeric-Literal-2 ] ... }
                                          {-}
                                          {*}
                                          {/}
           { LENGTH OF Data-Name-9                          }
           { BYTE-LENGTH OF Data-Name-10                    }
           { START OF Data-Name-10                          }
```

Format 5

```
01 Constant-Name-1

  CONSTANT AS { Nonnumeric-Literal-1                              } .
              { Numeric-Literal-1 [ {+} Numeric-Literal-2 ] ... }
                                   {-}
                                   {*}
                                   {/}
              { BYTE-LENGTH OF Data-Name-9                        }
              { LENGTH OF Data-Name-10                            }
              { START OF Data-Name-10                             }
```

Syntax rules

1. *Data-Name-1* or the FILLER clause, if specified, must immediately follow *Level-Number*.
2. If the REDEFINES clause is specified, the IDENTIFIED clause, if any, is ignored.
3. *Data-Name-1*, *Constant-Name-1* and *Condition-Name-1* are User-defined words, as defined in the Definitions section in the Preface of this document.
4. *Literal-1* is a Literal, as defined in the Definitions section in the Preface of this document. The category of *Literal-1* must match the category of the item being described.
5. *Nonnumeric-Literal-1* is a Nonnumeric Literal, as defined in the Definitions section in the Preface of this document.
6. *Numeric-Literal-1* and *Numeric-Literal-2* are Numeric Literals, as defined in the Definitions section in the Preface of this document.
7. *Literal-2*, *Literal-3* and *Literal-4* are Literals, as defined in the Definitions section in the Preface of this document.
8. *Nonnumeric-Literal-1*, *Numeric-Literal-1* and *Numeric-Literal-2* cannot be figurative constants.

General rules

Format 1

1. Level-Number must be 01 thru 49, or 77.
2. The SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO clauses must not be specified except for an elementary data item.

Format 3

1. Format 3 is used for each condition-name. Each condition-name requires a separate entry with level-number 88. Format 3 contains the name of the condition and the value, values, or range of values associated with the condition-name. The condition-name entries for a particular conditional variable must immediately follow the entry describing the item with which the condition-name is associated. A condition-name can be associated with any data description entry which contains a level-number except the following:
   a. Another condition-name.
   b. A level 66 item.
   c. A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).
   d. An index data item.
2. Multiple level 01 entries subordinate to any given level indicator, represent implicit redefinitions of the same area. See the FILE Section in the Data Division for details about level indicators.

3. *Data-Name-1* specifies the name of the data item being described. The key word FILLER may be used to specify a data item which is not referenced explicitly. If neither *Data-Name-1* nor FILLER are specified, FILLER is implied.
4. The VALUE clause and the condition-name itself are the only two clauses permitted in the entry. The characteristics of a condition-name are implicitly those of its conditional variable.
5. Wherever the THRU phrase is used, *Literal-2* must be less than *Literal-3*.

Format 4

1. The name of a constant can be used anywhere the corresponding Literal can be used.
2. If *Nonnumeric-Literal-1* is specified, the class and category of *Constant-Name-1* is the same as that of *Nonnumeric-Literal-1*.
3. If *Numeric-Literal-1* is specified, the class and category of *Constant-Name-1* is numeric.
4. If the BYTE-LENGTH phrase is specified, the class and category of *Constant-Name-1* is numeric. *Constant-Name-1* is an integer. The value of *Constant-Name-1* is determined as specified in the BYTE-LENGTH intrinsic function with the exception that when *Data-Name-9* is an occurs-depending group item, the maximum size of the data item is used.
5. If the LENGTH phrase is specified, the class and category of *Constant-Name-1* is numeric. *Constant-Name-1* is an integer. The value of *Constant-Name-1* is determined as specified in the LENGTH intrinsic function with the exception that when *Data-Name-10* is an occurs-depending group item, the maximum size of the data item is used.
6. If the START phrase is specified, the class and category of *Constant-Name-1* is numeric. *Constant-Name-1* is an integer. The value of *Constant-Name-1* is determined as the offset of *Data-Name-10*.

Format 5

1. Constant-name-1 may be used anywhere the corresponding Literal can be used.
2. If Nonnumeric-Literal-1 is specified, the class and category of *Constant-Name-1* is the same as that of Nonnumeric-Literal-1.
3. If Numeric-Literal-1 is specified, the class and category of *Constant-Name-1* is numeric.
4. If the BYTE-LENGTH phrase is specified, the class and category of *Constant-Name-1* is numeric. *Constant-Name-1* is an integer. The value of *Constant-Name-1* is determined as specified in the BYTE-LENGTH intrinsic function with the exception that when *Data-Name-1* is an occurs-depending group item, the maximum size of the data item is used.
5. If the LENGTH phrase is specified, the class and category of Constant-Name-1 is numeric. *Constant-Name-1* is an integer. The value of *Constant-Name-1* is determined as specified in the LENGTH intrinsic function with the exception that when *Data-Name-2* is an occurs-depending group item, the maximum size of the data item is used.

Level-Number

The level-number indicates the position of a data item within the hierarchical structure of a logical record. In addition, it is used to identify entries for working storage items, linkage items, screen items, condition-names, constant names and the RENAMES clause.

General format

Format 1

```
Level-Number
```

Format 2

```
66
```

Format 3

```
88
```

Format 4

```
78
```

Syntax rules

1. A level number is required as the first element in each data description entry.
2. Data description entries subordinate to an FD or SD entry must have level numbers with the values 01 through 49, 66, 77, 78 or 88.
3. Data description entries in the Working-Storage Section, Local-Storage Section and Linkage Section must have level numbers 01 through 49, 66, 77, 78 or 88.
4. Data description entries in the Screen Section must have level numbers 01 through 49, 66, 77, or 88.

General rules

1. The level number 01 identifies the first entry in each record description.
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 data items and can be used only as described by Format 1 of the data description entry.
   b. Level number 66 is assigned to identify `RENAMES` entries and can be used only as described by Format 2 of the data description entry.
   c. Level number 88 is assigned to entries which define condition names associated with a conditional variable and can be used only as described by Format 3 of the data description entry.
   d. Level number 78 is assigned to entries which define constant names representing literals and can be used only as described by Format 4 of the data description entry.
3. Multiple level 01 entries subordinate to any given level indicator represent implicit redefinitions of the same area. See the FILE Section in the Data Division for details about level indicators.

BLANK WHEN ZERO clause

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

General format

```
BLANK WHEN ZERO
```

Syntax rules

1. The BLANK WHEN ZERO clause can be specified only for an elementary item whose PICTURE is specified as numeric or numeric edited.
2. The numeric or numeric edited data description entry to which the BLANK WHEN ZERO clause applies must be described, either implicitly or explicitly, as USAGE IS DISPLAY.

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.

CONTROL clause

The CONTROL clause establishes a hierarchy of control breaks for the report.

General Format

```
{ CONTROL IS   }
{ CONTROLS ARE } { data-name-1 } ...
                  FINAL [ data-name-1 ] ...
```

Syntax rules

1. CONTROL and CONTROLS are synonyms.
2. Data-name-1 shall not be defined in the report section.
3. Data-name-1 may be qualified.
4. Data-name-1 may be reference modified. If it is, leftmost-position and length shall be integer literals.
5. The entry specified by data-name-1 shall not have a variable-occurrence entry subordinate to it.
6. Data-name-1 shall be unique in any given CONTROL clause. Two or more instances of data-name-1 in the same clause may, however, refer to the same physical data item or to overlapping data items.

General rules

1. The order of appearance of the operands of the CONTROL clause establishes the levels of the control hierarchy. The first data-name-1 is the major control; the next recurrence of data-name-1 is an intermediate control, etc. The last recurrence of data-name-1 is the minor control.
2. Specifying FINAL is equivalent to specifying a data item whose value does not change throughout the processing of the report. Therefore FINAL, if specified, is associated with the highest level in the hierarchy.
3. For each data-name-1 an internal data item, known as a prior control, is implicitly defined, having the same data description as the corresponding data item and a mechanism is established to save each control data item in the corresponding prior control and subsequently to compare these values to sense for control breaks.
4. Execution of the chronologically first GENERATE statement for a given report saves each current control data item in the corresponding prior control. Subsequent executions of any GENERATE statement for that report

automatically test the current value of each control data item, in order major to minor, for equality with the corresponding prior control. If a change of value in a control data item is detected, no further control data items are tested for the current GENERATE statement, and control break processing for that level and any lower levels is performed automatically.

5. If a control break has been detected during the execution of a GENERATE statement, the control data items are processed as follows:

   a. If any control footing is defined for the report, the current contents of control data items are saved and the corresponding prior controls are stored in the control data items before any control footing is printed. This ensures that any reference to a control data item in a control footing will always yield the value that the control data item had before the control break.

   b. If a USE BEFORE REPORTING declarative procedure is specified for the control footing, any reference to a control data item in this section will similarly yield the prior value.

   c. If the printing of the control footing causes a page advance and the page heading or page footing refers to a control data item, the value produced is similarly the prior value. When the printing of each control footing has been accomplished, the control data items are restored from the prior controls to their new current values, with these new current values remaining in the prior controls. All subsequent references to the control data items will obtain the new current values.

   d. If no control footing is defined for the report, the new current values of the control data items are stored in the corresponding prior controls.

6. When a TERMINATE statement is executed, if any control footing is defined for the report, the prior controls are stored in the control data items before each control footing is printed, as though a highest-level control break had been detected. The result is the same as that of a control break detected during the execution of a GENERATE statement.

7. Data-name-1 shall not reference a zero-length group item.

IS 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 in a run unit to every runtime element that describes the record as external.

General format

```
IS EXTERNAL
```

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 externalized name of the subject of the entry that includes the EXTERNAL clause shall not be the same as the externalized name of any other entry that includes the EXTERNAL clause.

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 within the run unit by any runtime element that describes the same record as external, subject to the following rules.
2. If the EXTERNAL clause is specified in a file description entry:

   a. the File connector associated with this file description entry is an external File connector; and

b. the data contained in all record description entries subordinate to that file description entry is external and may be accessed by any runtime element in the run unit that describes the same file and records as external, subject to the following rules.

3. Within a run unit, if two or more source elements describe the same external data record, each name that 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 bytes. 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.

IS EXTERNAL-FORM clause

The IS EXTERNAL-FORM clause associates a group item with HTML data using the CGI specification. It allows you to define input and output records for HTML forms. This syntax is supported for compatibility.

General Format

```
IS EXTERNAL-FORM [ IDENTIFIED BY template-file ]
IS IDENTIFIED BY external-name
```

IS IDENTIFIED clause

The `IS IDENTIFIED` clause specifies that a data item belongs to an XD entry, an XML structure, a JSON structure or HTTP parameters.

XD entries in FILE SECTION are supported for compatibility with other COBOLs.

XML structures in WORKING-STORAGE SECTION represent XML streams or XML files. Refer to the XMLStream Class (com.iscobol.rts.XMLStream) documentation for details about how to handle XML streams or XML files.

JSON structures in WORKING-STORAGE SECTION represent JSON streams. Refer to JSONStream Class (com.iscobol.rts.JSONStream) documentation for details about how to handle JSON streams.

HTTP parameters are used to exchange information between a COBOL program and a HTTP client in the EIS environment. Refer to isCOBOL Evolve: EIS for details about HTTP parameters.

General format

```
IS IDENTIFIED { BY    } Xml-Field [IS { ATTRIBUTE    }        ] [IS NULLABLE ]
              { USING }               { ELEMENT BASE64BINARY }
                                      { ELEMENT BOOLEAN      }
                                      { ELEMENT HEXBINARY    }
                                      { ELEMENT RAW          }
[ NAMESPACE IS NameSpace-Data ]
[ COUNT IN Count-Item ]
```

Syntax rules

1. *Xml-Field* is a Nonnumeric Literal or Data Item, as defined in the Definitions section in the Preface of this document. If *Xml-Field* is a data item, this clause must appear on a group item and *Xml-Field* must be defined directly within that group item.
2. *NameSpace-Data* is a Nonnumeric Literal or Data Item as defined in the Definitions section in the Preface of this document.

3. *Count-Item* is a Numeric Data Item as defined in the Definitions section in the Preface of this document. If not explicitly defined, the runtime defines it internally.

General rules

1. If the `IS IDENTIFIED` clause is specified for a subordinate item, the `IS IDENTIFIED` clause must be specified in the first entry of the record description the item belongs to.
2. When the `IS ATTRIBUTE` phrase is not specified,
   a. the data item identifies a field in the XML structure and *Xml-Field* represents the name of the field;
   b. the PICTURE clause cannot be specified, the value of the field in the stream of the file will be the content of subordinate data items;
   c. subordinate items cannot have the `IS IDENTIFIED` phrase specified, unless the `IS ATTRIBUTE` phrase is specified.
3. When the `IS ATTRIBUTE` phrase is specified,
   a. the data item must be a subordinate item of an item for which the `IS IDENTIFIED` clause without the `IS ATTRIBUTE` phrase is specified;
   b. the data item identifies an attribute of the field in the XML structure it belongs to;
   c. if the PICTURE clause is specified, the value of the attribute will be the content of the data item;
   d. if the PICTURE clause is not specified, the value of the attribute will be the content of the subordinate data items.
4. When `BASE64BINARY` is used, the data is coded in Base64 upon write and decoded in Base64 upon read.
5. When `BOOLEAN` is used, the data is considered a boolean value. This clause has effect only in JSON streams.
   During the write of a JSON stream, if the data item is numeric and contains a value greater than 0, the boolean value true is written, else the boolean value false is written; if the data item is alphanumeric and contains the word “true” (evaluated in case insensitive way), the boolean value true is written, else the boolean value false is written.
   During the read of a JSON stream, the boolean value true and the word “true” (evaluated in case insensitive way) are considered as true, any other value is considered as false. If the data item is numeric, it’s filled with 1 for true and 0 for false. If the data item is alphanumeric, it’s filled with the word “true” (lower case) for true and the word “false” (lower case) for false.
6. When `HEXBINARY` is used, the data is coded in hexadecimal upon write and decoded in hexadecimal upon read.
7. When `RAW` is used, special XML characters do not need to be escaped. This allows you to write everything in a tag, including new XML tags.
8. When `BASE64BINARY`, `BOOLEAN`, `HEXBINARY` or `RAW` are used, the data item may not be `USAGE CDATA`.
9. When `NULLABLE` is used, an empty value is replaced by the word 'null'. This clause has effect only in JSON streams.
10. The `NAMESPACE` clause specifies the value of the xmlns attribute of the field. The namespace URL must be provided. For example, the following XML element:

```
<MyElement xmlns:p="http://www.awebsite.com/app/elements/v1.0">
```

    is described as follows in the COBOL program:

```
01 MyElement identified by "MyElement"
              namespace "http://www.awebsite.com/app/elements/v1.0".
```

11. The namespace of an item is applied to all the sub items unless a different namespace is specified for them. If

the value of the NAMESPACE clause is an empty string, then no namespace is used.

12. *Count-Item* is not updated automatically and is evaluated by the classes that handle XML documents.
13. If *Xml-Field* is a data item, its content is evaluated at the moment XML structure is used by the runtime system.

IS SPECIAL-NAMES clause

The IS SPECIAL-NAMES clause provides an alternate way of assigning a data item to a special-name entry.

General format

```
IS SPECIAL-NAMES {CRT STATUS    }
                 {EVENT OBJECT  }
                 {EVENT SOURCE  }
                 {EVENT STATUS  }
                 {SCREEN CONTROL}
```

General rules

See the SPECIAL-NAMES Paragraph of the Environment Division for a detailed explanation of each clause.

IS THREAD-LOCAL clause

The THREAD-LOCAL clause specifies that there is a distinct copy of a data item for each thread entering a program.

General format

```
IS THREAD-LOCAL
```

Syntax rules

1. The EXTERNAL and GLOBAL clauses can't be specified along with THREAD-LOCAL.

General rules

1. If the THREAD-LOCAL clause is specified, a separate copy of the data item is created and set to its initial state for each new thread of execution that enters the program. The data item is only visible to the thread causing its creation. The data item is destroyed when the creating thread's execution terminates or when a CANCEL statement on the program is executed; otherwise, on subsequent calls to the program within that thread, the data item is in its last used state.
2. If the THREAD-LOCAL clause is not specified, the data item is shared by all threads entering the program.
3. In a single-threaded environment, the clause is ignored and the program behaves exactly as if the THREAD-LOCAL clause had not been specified.

IS TYPEDEF clause

The TYPEDEF clause defines a record as a programmer-defined type definition.

General format

```
IS TYPEDEF
```

Syntax rules

1. The TYPEDEF clause can be specified only in data description entries whose level-number is 01 or 77.
2. The following clauses cannot be specified along with TYPEDEF: EXTERNAL, GLOBAL, OCCURS, REDEFINES, VALUE.
3. If the TYPEDEF clause is specified for a group item, then subordinate items can be specified with OCCURS or REDEFINES clauses.
4. The VALUE clause cannot be specified either in the data descriptions specifying the TYPEDEF clause or in any subordinate item except for condition-names (88 level entries) within the TYPEDEF structure.
5. If the TYPEDEF clause is specified for a data description, then that same data description must include a data-name, that means it must not be specified with either an implicit or explicit FILLER clause.

General rules

1. The purpose of using the TYPEDEF clause is to create a programmer-defined usage or structure that can subsequently be referenced in the USAGE clause.
2. A record declared with the TYPEDEF clause does not allocate any storage.
3. A TYPEDEF defined in the Working-Storage Section of a FACTORY paragraph can be referenced by all the methods of the FACTORY's Procedure Division and all the methods of the OBJECT's Procedure Division. A TYPEDEF defined in the Working-Storage Section of a OBJECT paragraph can be referenced by all the methods of the OBJECT's Procedure Division. A TYPEDEF defined in the Working-Storage Section of a method can be referenced only by that method.

JUSTIFIED clause

The JUSTIFIED clause permits alternate positioning of data within a receiving data item.

General format

```
{JUSTIFIED} RIGHT
{JUST     }
```

Syntax rules

1. The JUSTIFIED clause can be specified only at the elementary item level.
2. JUST is an abbreviation for JUSTIFIED.
3. The JUSTIFIED clause cannot be specified for any data item described as numeric or for which editing is specified.
4. The JUSTIFIED clause must not be specified for an index data item.

General rules

1. When the receiving data item is described with the JUSTIFIED clause and the sending data item is larger than the receiving data item, the leftmost characters are truncated. When the receiving data item is described with the JUSTIFIED clause and it is larger than the sending data item, the data is aligned at the rightmost character position in the data item with space fill for the leftmost character positions.
2. When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply.

See Standard Alignment Rules.

NEXT GROUP clause

The NEXT GROUP clause specifies additional blank lines following the printing of the last line of a report group.

General format

```
NEXT GROUP IS integer-1 { PLUS } integer-2  NEXT PAGE [ WITH RESET ]
                        { + }
```

Syntax rules

1. Integer-1 specifies an absolute line number. Integer-2 specifies a relative vertical distance. Integer-1 and integer-2 shall not exceed the page limit, or 9999 if the report is not divided into pages. Integer-1 and integer-2 shall be unsigned.
2. PLUS and + are synonyms.
3. If the report is not divided into pages, only the relative form of the clause may be specified.
4. The NEXT GROUP clause shall not be specified in a page heading or report footing.
5. The NEXT PAGE phrase shall not be specified in a page footing.
6. If the absolute form is used, the following checks apply:
    a. If the current report group is a report heading, integer-1 shall be greater than the minimum last line number of the report group and less than the FIRST DETAIL integer.
    b. If the current report group is a body group, integer-1 shall lie between the FIRST DETAIL integer and the FOOTING integer, inclusive.
    c. If the current report group is a page footing, integer-1 shall be greater than the minimum last line number of the report group.
7. If the relative form is used, the following checks apply:
    a. If the current report group is a report heading, the minimum last line number of the report group plus integer-2 shall be less than the FIRST DETAIL integer.
    b. If the current report group is a page footing, the minimum last line number of the report group plus integer-2 shall not exceed the page limit.

General rules

1. The NEXT GROUP clause has no effect when it is specified in a control footing that is at a level other than the highest level at which the control break is detected.
2. The NEXT GROUP clause modifies the value of the current report's LINE-COUNTER after the printing of the last line, if any, of the report group in whose description the clause appears. The effect of this clause depends on the type of report group whose description contains the clause, as covered in the following general rules.
3. If the report group is a report heading, the effect of the absolute and relative forms is to increase the line number on which the immediately next report group is printed, namely the first body group when the report is not divided into pages, or the first page heading when the page heading consists only of relative lines. The effect on LINE-COUNTER in each case is as follows:
    a. If the NEXT GROUP clause is absolute, LINE-COUNTER is set equal to integer-1.
    b. If the NEXT GROUP clause is relative, integer-2 is added to LINE-COUNTER.

c. If NEXT GROUP NEXT PAGE is specified, the report heading is printed on the first page of the report as the only report group on that page and LINE-COUNTER is then set equal to zero.

4. If the report group is a body group:

   a. If the NEXT GROUP clause is absolute, a check is made as to whether LINE-COUNTER is less than integer-1. If so, LINE-COUNTER is set equal to integer-1. Otherwise, integer-1 is placed in a save location and LINE-COUNTER is set equal to the FOOTING integer, causing a page advance to take place just before any other non-dummy body group is printed for the report. The NEXT GROUP clause has no further effect on the current body group and will have no effect at all if a TERMINATE is next executed for the report. But it will affect the location of the next non-dummy body group, at the time that it is printed, in the following way:

      i. If the next body group begins with an absolute LINE clause without the NEXT PAGE phrase, the save location is moved to LINE-COUNTER and the page fit test is re-applied before the first line of the body group is printed.

      ii. If the next body group begins with an absolute LINE clause with the NEXT PAGE phrase, a page advance takes place, the save location is moved to LINE-COUNTER and a new page fit test and subsequent processing take place as for an identical report group without the NEXT PAGE phrase.

      iii. If the next body group contains only relative LINE clauses, its first line will be printed on the next line following the line number in the save location, unless this will result in some line of this body group being printed beyond its lower permitted limit. In the latter case, a second page advance takes place, resulting in a page devoid of body groups, and the next body group is printed on the following page with no reference to the save location.

   b. If the NEXT GROUP clause is relative and if the sum of integer-2 and LINE-COUNTER is less than the FOOTING integer, integer-2 is added to LINE-COUNTER; otherwise the FOOTING integer is moved to LINE-COUNTER.

   c. If NEXT GROUP NEXT PAGE is specified, the FOOTING integer is moved to LINE-COUNTER.

5. If the report group is a page footing, the NEXT GROUP clause affects any report footing defined in the current report using only relative LINE clauses, by changing LINE-COUNTER in the following ways:

   a. If the NEXT GROUP clause is absolute, LINE-COUNTER is set equal to integer-1.

   b. If the NEXT GROUP clause is relative, integer-2 is added to LINE-COUNTER.

6. If the WITH RESET phrase is present, the value of PAGE-COUNTER for the report is set to 1 (one) immediately after the page feed caused by the next page advance, chronologically between the printing of any page footing and the printing of any page heading. Whether the current group is a report heading or a body group, this phrase ensures that PAGE-COUNTER will be one, effective from the start of the next page (unless procedurally altered).

OCCURS clause

The OCCURS clause eliminates the need for separate entries for repeated data items and supplies information required for the application of subscripts.

General format

Format 1

```
OCCURS Integer-2 TIMES

  [ {ASCENDING } KEY IS {Data-Name-4} ... ] ...
    {DESCENDING}

  [ INDEXED BY {Index-Name} ... ]
```

Format 2

```
OCCURS FROM Integer-1 TO Integer-2 TIMES DEPENDING ON Data-Name-3

  [ {ASCENDING } KEY IS {Data-Name-4} ... ] ...
    {DESCENDING}

  [ INDEXED BY {Index-Name} ... ]
```

Format 3

```
OCCURS DYNAMIC [ CAPACITY IN Data-Name-5 ] [FROM Integer-1 ] [ TO Integer-
2 ] [ INITIALIZED ]

  [ {ASCENDING } KEY IS {Data-Name-4} ... ] ...
    {DESCENDING}

  [ INDEXED BY {Index-Name} ... ]
```

Format 4 (report writer):

```
OCCURS[ integer-1 TO] integer-2 TIMES [ DEPENDING ON data-name-1 ] [ STEP integer-3 ]
```

Format 5 (object oriented programming)

```
OCCURS Integer-2 TIMES
```

Syntax rules

1. The OCCURS clause must not be specified in a data description entry that has a variable occurrence data item subordinate to it.
2. *Data-Name-3* and *Data-Name-4* may be qualified.
3. The first specification of *Data-Name-4* must be the name of either the entry containing the OCCURS clause or an entry subordinate to the entry containing the OCCURS clause. Subsequent specification of *Data-Name-4* must be subordinate to the entry containing the OCCURS clause.
4. If *Data-Name-4* is subordinate to an alphanumeric group item or national group item that is subordinate to the entry containing the OCCURS clause, that group item shall not contain an OCCURS clause.
5. *Data-Name-4* must be specified without the subscripting normally required.
6. The data item identified by *Data-Name-4* shall not contain an OCCURS clause except when *Data-Name-4* is the subject of the entry.

7. *Data-Name-4* shall not reference a variable-length group.
8. Where both *Integer-1* and *Integer-2* are used, *Integer-1* must be greater than or equal to zero and *Integer-2* must be greater than *Integer-1*.
9. 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 sixteen.
10. *Data-Name-3* must describe an integer.
11. In Format 2, the data item defined by *Data-Name-3* must not occupy a character position within the range of the first character position defined by the data description entry containing the OCCURS clause and the last character position defined by the record description entry containing that OCCURS clause.
12. If the OCCURS clause is specified in a data description entry included in a record description entry containing the EXTERNAL clause, *Data-Name-3* shall reference a data item possessing the external attribute that is described in the same data division.
13. A data description entry that contains Format 2 of the OCCURS clause may only be followed, within that record description, by data description entries which are subordinate to it.
14. An INDEXED BY phrase is required if the subject of this entry, or an entry subordinate to this entry, is to be referenced by indexing.
15. *Index-name* must be a unique word within the program.
16. *Data-name-5* shall not be defined elsewhere in the source element. If qualifiers are required for uniqueness, it shall be treated as though implicitly defined at the same level as the entry containing the OCCURS clause.
17. *Data-name-5* shall not be referenced as a receiving item, except as the operand of a variable-table format SET Statement.
18. In Format 3, FROM and TO are treated as commentary.
19. When Format 3 occurs is included in group items, it doesn't alter the group item size and the offset the other sub-items in the group. Any COBOL statement performed on the group item will not consider the dynamic occurs inside.
20. When a Format 3 occurs is nested within another Format 3 OCCURS, the capacity of the nested OCCURS is set to the highest capacity. In order to retrieve the exact capacity of a nested Format 3 OCCURS, use a Format 15 SET statement.
21. Format 3 occurs items can be passed to a called program, but the called program must define the item in the same exact way (including the CAPACITY clause, if present) in order to receive these parameters correctly.
22. A Format 3 occurs cannot be mixed with other formats of occurs.
23. In Format 4, 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.
24. In Format 4, integer-1 shall be greater than or equal to zero and integer-2 shall be greater than integer-1.
25. In Format 4, data-name-1 shall describe an integer.
26. In Format 4, if the OCCURS clause is specified in an 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.
27. In Format 4, the TO and DEPENDING phrases shall either be both absent or both present.
28. In Format 4, the STEP phrase shall be specified if the entry:
    a. contains an absolute LINE clause, or
    b. has an entry with an absolute LINE clause subordinate to it, or
    c. contains an absolute COLUMN clause, or

d. is subordinate to an entry with a LINE clause and has an entry with an absolute COLUMN clause subordinate to it. In all other cases, the STEP phrase is optional.

29. In Format 4, the value of integer-3 shall be sufficient to prevent the overlapping of any line (in the case of vertical repetition) or column (in the case of horizontal repetition) of any two consecutive repetitions of the associated report item.
30. In Format 4, a report group description entry that contains an OCCURS clause with a DEPENDING phrase may be followed within that report group only by report group description entries that are subordinate to it.
31. A Format 5 occurs is used to create tables of OBJECT REFERENCE data items.
32. A Format 5 occurs can be used only for elementary items whose level number is either 01 or 77.

General rules

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. Format 3 defines a dynamic-capacity table.
3. The number of occurrences of the subject entry is defined as follows:

   a. In Format 1, the value of *Integer-2* represents the exact number of occurrences.

   b. In Format 2, the current value of the data item referenced by *Data-Name-3* represents the number of occurrences.

      This format specifies that the subject of this entry has a variable number of occurrences. The value of *Integer-2* represents the maximum number of occurrences and the value of *Integer-1* represents the minimum number of occurrences. This does not imply that the length of the subject of the entry is variable, but that the number of occurrences is variable.

      At the time the subject of 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-3* must fall within the range *Integer-1* through *Integer-2*.

   c. In Format 3, *Data-Name-5* contains the current capacity of the associated table. Its value is automatically updated as soon as the number of items in the associated table changes.
4. 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-3* 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-3* at the start of the operation will be used.

   b. If the data item referenced by *Data-Name-3* 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-3* 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.
5. When the KEY IS phrase is specified, the repeated data must be arranged in ascending or descending order according to the values contained in *Data-Name-4*. The ascending or descending order is determined according to the rules for the comparison of operands. The data-names are listed in their descending order of significance.
6. If Format 2 is specified in a record description entry and the associated file description or sort 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-3* of the OCCURS clause must be set to the number of occurrences to be written before the execution of any RELEASE, REWRITE, or WRITE Statement.
7. *Data-name-5* defines a numeric data item that contains the current capacity of the associated table. *Data-*

name-5 shall not be referenced as a receiving operand.

8. If `INITIALIZED` is specified, any unreferenced locations of each new entry added to the table are implicitly initialized.
9. The number of items stored in a Format 3 Occurs is stored in *Data-Name-5*.
10. In Format 4, if the OCCURS clause is written without any of the optional phrases, it causes the entry to define integer-2 distinct report items. The effect of the OCCURS clause on each repetition depends on the position of the entry containing the clause in the report group definition, as follows:

    a. If the entry also contains a relative COLUMN clause, each repetition behaves as though it had the same relative COLUMN clause.
    b. If the entry is a group entry having subordinate entries with relative COLUMN clauses, and being itself subordinate to an entry with a LINE clause, each repetition behaves as though it had the same subordinate entries with the same relative COLUMN clauses.
    c. If the entry also contains a relative LINE clause, each repetition behaves as though it had the same relative LINE clause.
    d. If the entry is a group entry having subordinate entries with relative LINE clauses, each repetition behaves as though it had the same subordinate entries with the same relative LINE clauses.
11. In Format 4, any PICTURE, USAGE, SIGN, VALUE, JUSTIFIED, BLANK WHEN ZERO, or GROUP INDICATE clauses have the same effect on each repetition as they would on a single data item without the OCCURS clause. This applies also to any SOURCE, SUM, or PRESENT WHEN clauses if no VARYING clause is present. If a VARYING clause is present, the action of these clauses may vary from one repetition to another. (See VARYING clause.)
12. In Format 4, the STEP phrase, if specified, defines the vertical or horizontal interval between successive occurrences of the associated report item after the first occurrence, as follows:

    a. If the entry contains a COLUMN clause, each successive occurrence is printed at a horizontal distance integer-3 columns to the right of the preceding occurrence.
    b. If the entry is a group entry having subordinate entries with COLUMN clauses and being itself subordinate to an entry with a LINE clause, printable items in each successive occurrence are positioned integer-5 columns to the right of the column they occupy in the preceding occurrence.
    c. If the entry contains a LINE clause, each successive occurrence is positioned integer-3 lines vertically beneath the preceding occurrence.
    d. If the entry is a group entry having subordinate entries with LINE clauses, report lines in successive occurrences are positioned integer-3 lines vertically beneath the line they occupy in the preceding occurrence.

    If no STEP phrase is specified, the vertical or horizontal interval between successive occurrences is defined by the relative LINE or COLUMN numbers, respectively, specified in the corresponding report section entries.
13. In Format 4, if the DEPENDING phrase is specified, the value of data-name-1 is evaluated just before the processing of the first LINE clause of the report group. If the value of data-name-1 is not in the range integer-1 to (integer-2 - 1), the report group is processed as though the OCCURS clause had been written without the TO and DEPENDING phrases. If the value of data-name-1 is in the range integer-1 to (integer-2 - 1), the OCCURS clause has the same effect as an OCCURS clause with no TO or DEPENDING phrases and with an integer-2 equal to the current value of data-name-1.
14. If you are using nested OCCURS DEPENDING clauses, you must use -cod1 compiler option.

PICTURE clause

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

Format 1

```
{ PICTURE } IS Character-String
{ PIC     }
```

Format 2

```
{ PICTURE } IS { X } ANY LENGTH
{ PIC     }    { N }
               { G }
```

Format 3

```
{ PICTURE } IS Character-String VARYING
{ PIC     }
```

Syntax rules

1. The PICTURE clause can be specified only at the elementary item level.
2. *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, G, N, 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 100.
5. The PICTURE clause must be specified for every elementary item, except those specifying *Usage-String-2*.
6. The words PICTURE and PIC are equivalent.
7. The asterisk when used as the zero suppression symbol and the clause BLANK WHEN ZERO may not appear in the same entry
8. Format 3 Character-String can contain only X symbols.

General rules

Format 1

1. Categories of data that can be described with a PICTURE clause are: alphabetic, numeric, alphanumeric, national, alphanumeric edited, numeric edited and national edited.
2. To define an item as alphabetic:
   a. Its PICTURE character-string can contain only the symbol 'A'; and
   b. Its content, when represented in standard data format, must be one or more alphabetic characters
3. To define an item as numeric:
   a. Its PICTURE character-string can contain only the symbols '9', 'P', 'S', and 'V'. The number of digit positions that can be described by the PICTURE character-string must range from 1 to 31 inclusive; and
   b. If unsigned, its content when represented in standard data format must be one or more numeric characters; if signed, the item may also contain a '+ ', '-', or other representation of an operational sign. See the SIGN clause for further details.

4. To define an item as alphanumeric:

    a. Its PICTURE character-string is restricted to certain combinations of the symbols 'A', 'G', 'X', '9', and the item is treated as if the character-string contained all 'X's. A PICTURE character-string which contains all 'A's or all '9's does not define an alphanumeric item, and;

    b. Its content when represented in standard data format must be one or more characters in the computer's character set.

5. To define an item as national:

    a. Its PICTURE character-string can contain one or more occurrences of the symbol 'N'.

6. To define an item as alphanumeric edited:

    a. Its PICTURE character-string is restricted to certain combinations of the following symbols: 'A', 'G', 'X', '9', 'B', '0', and '/'; and must contain at least one 'A', 'G' or 'X' and must contain at least one 'B' or '0' (zero) or '/' (slant).

    b. Its content when represented in standard data format must be two or more characters in the computer's character set.

7. To define an item as 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

        i. The number of digit positions that can be represented in the PICTURE character-string must range from 1 to 36 inclusive; and

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

    b. The content of each of the character positions must be consistent with the corresponding PICTURE symbol.

8. To define an item as national edited:

    a. Its PICTURE character-string must contain at least one symbol 'N'; and

    b. At least one '0', 'B' or '/' (slant) symbol.

9. The size of an elementary item, where size means the number of character positions occupied by the elementary item in standard data format, is determined by the number of allowable symbols that represent character positions. An unsigned nonzero integer which is enclosed in parentheses following the symbols 'A', 'G', ',', 'X', '9', 'P' 'Z', '*', 'B', '/', '0', '+', '-', 'S', 'V', or the currency symbol indicates the number of consecutive occurrences of the symbol. Note that the following symbols may appear only once in a given PICTURE: 'S', 'V, '.', 'CR', and 'DB'.

10. The functions of the symbols used to describe an elementary item are explain as follows:

    - **A** - Each 'A' in the character-string represents a character position which can contain only an alphabetic character and is counted in the size of the item.
    - **B** - Each 'B' in the character-string represents a character position into which the space character will be inserted and is counted in the size of the item.
    - **G** - Each "G" represents a character position which can contain only a double-byte character set (DBCS) character or a DBCS space. Each symbol 'G' is counted in the size of the item.
    - **N** - Each symbol 'N' represents a national character position that shall contain a character from the computer's national character set. Each symbol 'N' is counted in the size of the item.
    - **P** - Each 'P' in the character-string indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character 'P' is not counted in the size of the data

item. Scaling position characters are counted in determining the maximum number of digit positions (36) in numeric edited items or numeric items. The scaling position character 'P' can appear only as a continuous string of 'P's in the leftmost or rightmost digit positions within a PICTURE character-string; since the scaling position character 'P' implies an assumed decimal point (to the left of 'P's if 'P's are leftmost PICTURE symbols and to the right if 'P's are rightmost PICTURE symbols), the assumed decimal point symbol 'V' is redundant as either the leftmost or rightmost character within such a PICTURE description. The symbol 'P' and the insertion symbol '.' (period) cannot both occur in the same PICTURE character-string.

In certain operations that reference a data item whose PICTURE character-string contains the symbol 'P', the algebraic value of the data item is used rather than the actual 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:

i. Any operation requiring a numeric sending operand.

ii. A MOVE Statement where the sending operand is numeric and its PICTURE character-string contains the symbol 'P'.

iii. 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.

iv. 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 must be written as the leftmost character in the PICTURE. The 'S' is not counted in determining the size (in terms of standard data format characters) of the elementary item unless the entry is subject to a SIGN clause which specifies the optional SEPARATE CHARACTER phrase.
- **V** - The 'V' is used in a character-string to indicate the location of the assumed decimal point and may only appear once in a character-string. The 'V' does not represent a character position and therefore is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string representing a digit position or scaling position, the 'V' is redundant.
- **X** - Each 'X' in the character-string is used to represent a character position which contains any allowable character from the computer's character set and is counted in the size of the item.
- **Z** - Each 'Z' in a character-string may only be used to represent the leftmost leading numeric character positions which will be replaced by a space character when the content of that character position is a leading zero. Each 'Z' is counted in the size of the item.
- **9** - Each '9' in the character-string represents a digit position which contains a numeric character and is counted in the size of the item.
- **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 which represents the decimal point for alignment purposes and, in addition, represents a character position into which the character '.' will be inserted. The character '.' is counted in the size of the

item. For a given program the functions of the period and comma are exchanged if the clause `DECIMAL-POINT IS COMMA` is stated in the `SPECIAL-NAMES` paragraph. In this exchange the rules for the period apply to the comma and the rules for the comma apply to the period wherever they appear in a `PICTURE` clause.

- **+ - CR DB** - These symbols are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in any one character-string and each character used in the symbol is counted in determining the size of the data item.
- ***** - Each '*' (asterisk) in the character-string represents a leading numeric character position into which an asterisk will be placed when the content of that position is a leading zero. Each '*' is counted in the size of the item.
- **cs** - The currency symbol in the character-string represents a character position into which a currency symbol is to be placed. The currency symbol in a character-string is represented by either the currency sign or by the single character specified in the `CURRENCY SIGN` clause in the `SPECIAL-NAMES` paragraph. The currency symbol is counted in the size of the item.

11. The PICTURE clause may be omitted for an elementary item when an alphanumeric or national literal is specified in the VALUE clause. A PICTURE clause is implied as follows:
    - if the literal is national, 'PICTURE N(*length*)'.
    - if the literal is alphanumeric, 'PICTURE X(*length*)'
      where *length* is the length of the literal.

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:

   a. Simple insertion
   b. Special insertion
   c. Fixed insertion
   d. Floating insertion

   There are two types of suppression and replacement editing:

   a. Zero suppression and replacement with spaces
   b. Zero suppression and replacement with asterisks

2. The type of editing which may be performed upon an item is dependent upon the category to which the item belongs. The following table specifies which type of editing may be performed upon a given category:

Category	Type of Editing
Alphabetic	None
Numeric	None
Alphanumeric	None
National	None
Alphanumeric edited	Simple insertion '0', 'B1, and '/
Numeric edited	All, subject to rules in rule 3 below

National edited	Simple insertion

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 characters. The insertion characters are counted in the size of the item and represent the position in the item into which the character will be inserted. If the insertion character ',' (comma) is the last symbol in the PICTURE Character-String, the PICTURE clause must be the last clause of the data description entry and must be immediately followed by the separator period. This results in the combination of ',.' appearing in the data description entry, or, if the DECIMAL POINT IS COMMA clause is used, in two consecutive periods.
5. Special insertion editing. The '.' (period) is used as the insertion character. In addition to being an insertion character it also represents the decimal point for alignment purposes. The insertion character used for the actual decimal point is counted in the size of the item.The use of the assumed decimal point, represented by the symbol 'V' and the actual decimal point, represented by the insertion character, in the same PICTURE character-string is disallowed. If the insertion character is the last symbol in the PICTURE character-string, the PICTURE clause must be the last clause of that data description entry and must be immediately followed by the separator period. This results in two consecutive periods appearing in the data description entry, or 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 shown in the character-string.
6. Fixed insertion editing. The currency symbol and the editing sign control symbols '+', '-', 'CR', 'DB' are the insertion characters. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character-string. When the symbols 'CR' or 'DB' are used they represent two character positions in determining the size of the item and they must represent the rightmost character positions that are counted in the size of the item. If these character positions contain the symbols 'CR' or 'DB', the uppercase letters are the insertion characters. The symbol '+' or '-', when used, must be either the leftmost or rightmost character position to be counted in the size of the item. The currency symbol must be the leftmost character position to be counted in the size of the item except that it can be preceded by either a '+' or a '-' symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the PICTURE character-string. Editing sign control symbols produce the following results depending upon the value of the data item:

Editing symbol in picture Character-String	Result Data item in 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 characters and as such are mutually exclusive in a given PICTURE character-string.

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

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

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

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

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

If all numeric character positions in the PICTURE character-string are represented by the insertion character, at least one of the insertion characters must be to the left of the decimal point.

When the floating insertion character is the editing control symbol '+' or '-', the character inserted depends upon the value of the data item:

Editing symbol in picture Character-String	Result Data item in positive or zero	Data item negative
+	+	-
-	Space	-

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

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of nonfloating insertion characters being edited into the receiving data item, plus one for the floating insertion character. If truncation does occur, the value of the data that is used for editing is the value after truncation. See Standard Alignment Rules.

8. Zero suppression editing. The suppression of leading zeros in numeric character positions is indicated by the use of the alphabetic character 'Z' or the character '*' (asterisk) as suppression symbols in a PICTURE character-string. These symbols are mutually exclusive in a given PICTURE character-string. Each suppression symbol is counted in determining the size of the item. If 'Z' is used the replacement character will be the space and if the asterisk is used, the replacement character will be '*'.

   Zero suppression and replacement is indicated in a PICTURE character-string by using a string of one or more of the allowable symbols to represent leading numeric character positions which are to be replaced when the associated character position in the data contains a leading zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string.

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

If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates 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 character positions in the PICTURE character-string are represented by suppression symbols and the value of the data is not zero the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero and the suppression symbol is 'Z', the entire data item, including any editing characters, is spaces. If the value is zero and the suppression symbol is '*', the entire data item, including any insertion editing 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 characters, are mutually exclusive within a given character-string.

Precedence rules

The chart below shows the order of precedence when using characters as symbols in a character-string. An 'X' at an intersection indicates that the symbol(s) at the top of the column may precede (but not necessarily immediately), in a given character-string, the symbol(s) at the left of the row. Arguments appearing in braces { } indicate that the symbols are mutually exclusive.The currency symbol is indicated by the symbol 'cs'.

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

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

First Symbol		Non-Floating Insertion Symbols							Floating Insertion Symbols						Other Symbols						
Second Symbol		{B} {0} {/}	,	.	{+} {-}	{+} {-}	{CR} {DB}	cs	{Z} {*}	{Z} {*}	{+} {-}	{+} {-}	cs	cs	9	[A] [X]	S	V	P	P	N
Non-Floating Insertion Symbols	{B|0|/}	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x	x
	,	x	x	x	x			x	x	x	x	x	x	x				x		x	
	.	x	x		x			x	x		x		x		x						
	{+|-}																				
	{+|-}	x	x	x				x	x	x			x	x	x			x	x	x	
	{CR|DB}	x	x	x				x	x	x			x	x	x			x	x	x	
	cs				x																
Floating Insertion Symbols	{Z|*}	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	
	cs	x	x		x								x								
	cs	x	x	x	x								x	x				x		x	
Other Symbols	9	x	x	x	x			x	x		x		x		x	x	x	x		x	
	[A|X]	x													x	x					
	S																				
	V	x	x		x			x	x		x		x		x		x		x		
	P	x	x		x			x	x		x		x		x		x		x		
	P				x			x									x	x		x	
	N	x																			x

Format 2

1. ANY LENGTH items are always alphanumeric.
2. ANY LENGTH items are zero bytes in size when they're initialized.
3. The size of an ANY LENGTH length item's size corresponds to the size of the value that is stored in it.
4. When referenced over their size, ANY LENGTH items are automatically resized to make the referenced area exist.
5. When ANY LENGTH items are included in group items, they don't alter neither the group item size nor the offset the other sub-items. COBOL statements that use the group item as a buffer of bytes without caring about the item structure will not consider the ANY LENGTH items inside. It is good practice to use ANY LENGTH items as elementary items.
6. When an ANY LENGTH item is used in the Linkage Section, that item can receive any alphanumeric parameter: a constant string, a fixed length alphanumeric item or another item with picture ANY LENGTH.
7. If a caller program passes an ANY LENGTH item and the callee receives it in a fixed length alphanumeric item, the callee may not try to access the parameter over the boundaries dictated by the current size of the ANY LENGTH item, otherwise an IndexOutOfBounds error occurs.
8. Setting an ANY LENGTH item to the figurative constant SPACE initializes the data item making it zero bytes in size. This rule applies to MOVE and SET statements as well as the VALUE clause. Moving space characters to an ANY LENGTH item instead resizes the item to host those space characters. The below code snippet

summarizes this rule:

```
      working-storage section.
       77 wrk-item pic x any length value space.

       procedure division.
       main-logic.
      *At program start, wrk-item is 0 bytes in size and hosts no data
           move "   " to wrk-item.
      *after the above MOVE, wrk-item is 3 bytes in size and hosts 3 spaces
           move space to wrk-item.
      *after the above MOVE, wrk-item is 0 bytes in size and hosts no data
```

9. Setting an ANY LENGTH item to the figurative constants ZERO, LOW-VALUE, HIGH-VALUE, ALL *literal* or ALL *symbolic character* causes all the characters in the item to be replaced with zeros, low values, high values, literals or symbolic characters respectively. It the ANY LENGTH item is zero bytes in size, the setting has no effect.
10. The proper way to resize ANY LENGTH items to a specific size is by using a format 2 INITIALIZE statement.

Format 3

1. VARYING items are used in ESQL programs to handle VARCHAR fields. The item is internally translated into:

```
01  Data-Name-1.
    03  Data-Name-1-arr pic X(n).
    03  Data-Name-1-len pic S9(4) COMP-5.
```

   Where n is the number of digits specified in Character-String. The first field contains the text of the VARCHAR field, the second fields contains the length of the value for the VARCHAR field.
2. VARYING items shouldn't be used to intercept NULL values. The length parameter is set to unpredictable values in this case.

   For more information see Mapping a VARCHAR field to a COBOL group data item.

PRESENT WHEN clause

1. The PRESENT WHEN clause specifies a condition under which a report section entry will be processed.
2. The PRESENT WHEN clause also enables conditional selection of data description entries by the VALIDATE statement.

General format

Format 1 (report-writer):

```
PRESENT WHEN Condition-1
```

Format 2 (validation):

```
PRESENT WHEN Condition-1
```

Syntax rules

Format 2

1. The PRESENT WHEN clause shall not be specified for a strongly-typed group item or any item subordinate to a strongly-typed group item.

General rules

Formats 1 and 2

1. If the PRESENT WHEN clause is specified in a report group description entry, the general rules for format 1 apply, otherwise, the general rules for format 2 apply.

Format 1

2. If a report group contains any entries that have a PRESENT WHEN clause, condition-1 of each PRESENT WHEN clause is evaluated before the processing of any LINE clauses for the report group. The effect of the PRESENT WHEN clause depends on the value of condition-1 as follows:

    a. If condition-1 is true, the corresponding data item is declared to be present and the PRESENT WHEN clause does not affect the processing for this instance of the report group.

    b. If condition-1 is false, the corresponding data item is declared to be absent and the effect on processing is as though the entry were omitted from the description of the report group. If the data description entry is not an elementary entry, all its subordinate data items are also declared to be absent, irrespective of any PRESENT WHEN clauses they may also contain. Furthermore, if the entry is a level-01 entry, the effect on processing is as though the entire report group description were omitted.

3. Within a report group description, any PRESENT WHEN clauses are taken into account when assessing the validity of the arrangement of LINE and COLUMN clauses, the manner in which the report group will be printed and the effect of sum counters, as follows:

    a. The rules for positioning the first line of the report group ignore any LINE clauses specified at the start of the report group where the LINE clauses are associated with absent data items. (See LINE clause.)

    b. The rules forbidding overlap of absolute lines in the report group are not applied to lines associated with absent data items. (See LINE clause.)

    c. The rules preventing trailing relative lines in the report group from exceeding the report group's lower limit are not applied to lines associated with absent data items. (See LINE clause.)

    d. The page fit test for body groups disregards all lines associated with absent data items. (See LINE clause.)

    e. The rules forbidding overlap of absolute printable items in a report line are not applied to items associated with absent data items. (See COLUMN clause, general rule 4.)

f. The rules preventing trailing relative printable items in the line from exceeding the page width are not applied to items associated with absent data items. (See COLUMN clause.)

g. If an entry with a SUM clause is associated with an absent data item, the sum counter is not printed and is not reset to zero.

Format 2

4. The PRESENT WHEN clause takes effect during the execution of a VALIDATE statement that directly or indirectly references the subject of the entry.
5. Condition-2 is evaluated at the beginning of the execution of the format validation stage, with the following two possible results:

   a. If condition-2 is true, the data item that is the subject of the entry is processed during further execution of the VALIDATE statement.

   b. If condition-2 is false, the data item that is the subject of the entry and all data items subordinate to it are not processed at this and all subsequent stages of the execution of the VALIDATE statement.

      NOTE - If condition-2 is false, the contents of the data item will not be checked unless the data item is redefined.
6. Condition-2 shall not reference any data item that is, or shares any storage with, an operand of a DESTINATION clause appearing later in the description of a data item referred to by the same VALIDATE statement.

PROPERTY clause

The `PROPERTY` clause allows to define a property in Object Oriented Programming.

General format

```
PROPERTY
```

Syntax rules

1. The `PROPERTY` clause can be specified only in the Working-Storage Section of a factory definition or an object definition.
2. The `PROPERTY` clause must not be specified for data items subordinate to an OCCURS clause.
3. The `PROPERTY` clause can be specified only for an elementary item whose name does not require qualification for uniqueness of reference.
4. The data-name for the subject of the entry must not be the same as a property-name defined in a superclass.

General rules

1. the PROPERTY clause causes a GET method and a SET method to be defined for the containing factory or object. The implicit definition of these method is as follows:

```
METHOD-ID. GET PROPERTY data-name.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-data-name data-description.
PROCEDURE DIVISION RETURNING LS-data-name.
par-name.
    MOVE data-name TO WS-data-name
    EXIT METHOD.
END METHOD.

METHOD-ID. SET PROPERTY data-name.
DATA DIVISION.
LINKAGE SECTION.
01 LS-data-name data-description.
PROCEDURE DIVISION USING LS-data-name.
par-name.
    MOVE LS-data-name TO data-name
    EXIT METHOD.
END METHOD.
```

These methods can be overridden in order to use custom code.

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] [ REDEFINES Data-Name-2 ]
             [FILLER     ]
```

NOTE - Level-Number, Data-Name-1, and FILLER are shown in the above format to improve clarity. Level-Number, Data-Name-1, and FILLER are not part of the REDEFINES clause.

Syntax rules

1. *Level-Number* of *Data-Name-2* and *Level-Number* of the item being described must be identical.
2. This clause must not be used in level 01 entries in the File Section.
3. The data description entry for *Data-Name-2* cannot contain an OCCURS clause. However, *Data-Name-2* may be subordinate to an item whose data description entry contains an OCCURS clause. In this case, the reference to *Data-Name-2* may not be subscripted. Neither the original definition nor the redefinition can include a variable occurrence data item.
4. The number of character positions described by *Data-Name-1* need not be the same as the number of character positions in the subject of the REDEFINES clause. The compiler generates a warning, however, if the number of character positions is greater in the subject of the REDEFINES clause than in *Data-Name-1*. By default the warning is returned only if *Data-Name-1* and *Data-Name-2* are not group items. In order to extend the check on REDEFINES lenght also to group items, use the -wr compiler option.
5. *Data-Name-2* must not be qualified even if it is not unique since no ambiguity of reference exists in this case

because of the required placement of the REDEFINES clause within the source program.

6. The entries giving the new description of the character positions must not contain any VALUE clauses, except in subordinate condition-name entries.
7. No entry having a level number numerically lower than the level number of *Data-Name-2* and the subject of the entry may occur between the data description entries of *Data-Name-2* and the subject of the entry.
8. The entries giving the new descriptions of the character positions must follow the entries defining the area of *Data-Name-2*, without intervening entries that define new character positions.
9. *Data-Name-2* may be subordinate to an entry which contains a REDEFINES clause.

General rules

1. Storage allocation starts at *Data-Name-2* and continues over a storage area sufficient to contain the number of character positions in the data item referenced by *Data-Name-1* or the FILLER clause.
2. When the same character position is defined by more than one data description entry, the data-name associated with any of those data description entries can be used to reference that character position.

REPORT clause

The REPORT clause identifies the reports that may be written to a report file.

General format

```
{ REPORT  IS  }  { report-name-1 } ...
{ REPORTS ARE }
```

Syntax rules

1. Each report-name-1 shall be the subject of a report description entry in the report section of the same source element. The order of appearance of each report-name-1 is not significant.
2. Each report-name-1 may appear in only one REPORT clause.
3. The subject of a file description entry that specifies a REPORT clause may be referenced in the procedure division only by the USE statement, the CLOSE statement, or the OPEN statement with the OUTPUT or EXTEND phrase.

General rules

1. The presence of more than one report-name-1 indicates that more than one report may be written to the file.
2. After execution of an INITIATE statement and before the execution of a TERMINATE statement for the same report, no OPEN or CLOSE statements shall be executed that reference the report file.

RENAMES clause

The RENAMES clause permits alternative, possibly overlapping, groupings of elementary items.

General format

```
66 Data-Name-6 RENAMES Data-Name-7 [ {THROUGH} Data-Name-8 ] .
                                     {THRU   }
```

NOTE - The level number 66 and Data-Name-6 are shown in the above format to improve clarity. The level number 66 and Data-Name-6 are not part of the RENAMES clause.

Syntax rules

1. Any number of RENAMES entries may be written for a logical record.
2. All RENAMES entries referring to data items within a given logical record must immediately follow the last data description entry of the associated record description entry.
3. *Data-Name-6* cannot be used as a qualifier, and can only be qualified by the names of the associated level 01, FD, or SD entries. Neither *Data-Name-7* nor *Data-Name-8* 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-7* and *Data-Name-8* must be names of elementary items or groups of elementary items in the same logical record, and cannot be the same data-name. A 66 level entry cannot rename another 66 level entry nor can it rename a 77, 88, or 01 level entry.
5. *Data-Name-7* and *Data-Name-8* may be qualified.
6. None of the items within the range, including *Data-Name-7* and *Data-Name-8*, if specified, can be variable occurrence data items.
7. The words THROUGH and THRU are equivalent.
8. The beginning of the area described by *Data-Name-8* must not be to the left of the beginning of the area described by *Data-Name-7*. The end of the area described by *Data-Name-8* must be to the right of the end of the area described by *Data-Name-7*. *Data-Name-8*, therefore, cannot be subordinate to *Data-Name-7*.

General rules

1. When *Data-Name-8* is specified, *Data-Name-6* is a group item which includes all elementary items starting with *Data-Name-7* (if *Data-Name-7* is an elementary item) or the first elementary item in *Data-Name-7* (if *Data-Name-7* is a group item), and concluding with *Data-Name-8* (if *Data-Name-8* is an elementary item) or the last elementary item in *Data-Name-8* (if *Data-Name-8* is a group item).
2. When *Data-Name-8* is not specified, all of the data attributes of *Data-Name-7* become the data attributes for *Data-Name-6*.

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 } [ SEPARATE CHARACTER ]
            {TRAILING}
```

Syntax rules

1. The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character 'S', or a group item containing at least one such numeric data description entry.
2. The numeric data description entries to which the SIGN clause applies must be described, implicitly or explicitly, as USAGE IS DISPLAY.

General rules

1. The optional SIGN clause, if present, specifies the position and the mode of representation of the operational sign for the numeric data description entry to which it applies, or for each numeric data description entry subordinate to the group to which it applies. The SIGN clause applies only to numeric data description entries whose PICTURE contains the character 'S'; the 'S' indicates the presence of, but neither the representation nor, necessarily, the position of the operational sign.

2. If a SIGN clause is specified in a group item subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in the subordinate group item takes precedence for that subordinate group item.
3. If a SIGN clause is specified in an elementary numeric data description entry subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in the subordinate elementary numeric data description entry takes precedence for that elementary numeric data item.
4. A numeric data description entry whose PICTURE contains the character 'S', but to which no optional SIGN clause applies, has an operational sign, but neither the representation, nor, necessarily, the position of the operational sign is specified by the character 'S'. General rules 5 through 7 do not apply to such signed numeric data items.
5. If the optional SEPARATE CHARACTER phrase is not present, then:
    a. The operational sign will be presumed to be associated with the leading (or, respectively, trailing) digit position of the elementary numeric data item.
    b. The letter 'S' in a PICTURE character-string is not counted in determining the size of the item (in terms of standard data format characters).
6. If the optional SEPARATE CHARACTER phrase is present, then:
    a. The operational sign will be presumed to be the leading (or, respectively, trailing) character position of the elementary numeric data item; this character position is not a digit position.
    b. The letter 'S' in a PICTURE character-string is counted in determining the size of the item (in terms of standard data format characters).
    c. The operational signs for positive and negative are the standard data format characters '+' and '-', respectively.
7. Every numeric data description entry whose PICTURE contains the character 'S' is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

SUM clause

The SUM clause specifies one or more data items that are to be totalled to provide the value of the associated elementary report item.

General format

```
SUM OF { data-name-1  }       [ UPON { data-name-2 } ... ]
       { identifier-1 }
       { arithmetic-expression-1}

RESET ON data-name-3 FINAL  [ ROUNDED ]
```

Syntax rules

1. Each data-name-1, identifier-1 or arithmetic-expression-1 is an addend. The whole clause is referred to as a SUM clause even though the SUM keyword may appear more than once.
2. The category of the subject of the entry shall be valid as the category of a receiving operand in a MOVE statement for a sending operand of the category numeric.
3. Data-name-1 shall be the name of a numeric data item in the report section. It may be qualified. If it is associated with an OCCURS clause, it shall be specified without the subscripting normally required. When data-name-1 is specified, the following rules also apply:
    a. The UPON phrase shall not be specified.

b. If data-name-1 is specified in the same report group description as the subject of the entry, data-name-1 shall be a repeating item, as defined in Report group description entry, general rule 3, and subject to at least one more level of repetition than the subject of the entry.

c. If data-name-1 is specified in a different report group description than the subject of the entry, data-name-1 either shall not be a repeating item or shall be a repeating item that is subject to at least the same number of levels of repetition as the subject of the entry.

d. The maximum number of repetitions of data-name-1 and the subject of the entry shall be equal at each corresponding level taken in order beginning with the lowest level of nesting. Levels superordinate to those corresponding levels may specify any number of repetitions.

e. Any chain of reference shall terminate at an entry that does not contain a SUM clause referring to a dataname-1 defined in the report section.

f. If data-name-1 specifies an entry in a report group description other than the current report group description, only the following combinations of report types of each report group are permitted: The current report group may be a control footing and data-name-1 may be defined in a detail or in a control footing associated with a lower level of control. The current report group may be a detail and data-name-1 may be defined in a different detail. The current report group may be a report footing and data-name-1 may be defined in any other report group other than a report heading. The current report group may be a page footing and data-name-1 may be defined in any body group.

g. If data-name-1 specifies an entry in a different report description, there are no restrictions on the combinations of report types of each report group.

4. If the addend is identifier-1, it shall specify a numeric data item not defined in the report section.
5. If the addend is arithmetic-expression-1, any identifiers it contains may reference entries in any section of the data division other than the report section.
6. Data-name-2 shall be the name of a detail. It may be qualified by a report-name.
7. Data-name-3 may be qualified and reference modified. Data-name-3 or FINAL shall be an operand of the CONTROL clause of the current report description. If data-name-3 is reference modified, leftmost-position and length shall be integer literals. If the current report group is a control footing, its level of control shall be a lower level than that of data-name-3.

General rules

1. Each entry containing a SUM clause establishes an independent sum counter and size error indicator. The sum counter is an internal register that behaves as a data item of the category numeric. The number of decimal digits in the sum counter, both integral and fractional, is derived from the corresponding number of digits, excluding insertion editing characters, in the PICTURE clause of the entry containing the SUM clause. The sum counter is signed, whether or not the corresponding PICTURE clause has an operational sign.
2. The sum counter is set to zero and its associated size error indicator is unset when the INITIATE statement for the current report is executed. Subsequently, the sum counter is reset to zero and the size error indicator is unset at the end of the processing of the report group in which it is printed or, if the RESET phrase is specified, at the end of the processing of the control footing for the specified level of control. If no such control footing is defined, it is assumed to be present and to consist of a 01-level entry alone.
3. The current content of each addend is implicitly added into the sum counter during execution of GENERATE and TERMINATE statements at specific times defined below. The adding is consistent with the general rules of the ADD statement with the ON SIZE ERROR phrase or, in the case of an arithmetic expression, the COMPUTE statement with the ON SIZE ERROR phrase. Any algebraic sign of the addend is taken into account. Each addition is tested for size error; if a size error occurs, the associated size error indicator is set.
4. If the entry also contains a COLUMN clause, the sum counter acts as a source data item. If the associated size error indicator is not set, the content of the sum counter is moved, according to the general rules of the MOVE statement, to the printable item for printing. If the associated size error indicator is set, an EC-REPORT-SUM-SIZE exception condition is set to exist and the printable item is filled with spaces.
5. If a data-name immediately follows the level number in the entry containing the SUM clause, the data-name is the name of the sum counter, not the name of the associated printable item, if any.

6. If data-name-1 specifies an item whose entry contains a SUM clause, the value added is that of the corresponding sum counter. The additions necessary to compute its value are completed before the adding of the operand into the current sum counter. If data-name-1 specifies an item whose entry has a SOURCE or VALUE clause, the value added is that of the operand of the SOURCE or VALUE clause.
7. The times at which addition takes place are defined as follows:

    a. If data-name-1 is the name of an entry in a different report group description, adding takes place when the report group description containing data-name-1 is processed.

    b. If data-name-1 is the name of an entry in the current report group description, adding takes place during the processing of the current report group before any of the report group's lines are printed.

    c. If the addend is identifier-1 or arithmetic-expression-1, adding takes place either:

        i. if no UPON phrase is specified, whenever any GENERATE statement is executed for the current report or any detail defined for the current report, or

        ii. If an UPON phrase is specified, whenever any GENERATE statement is executed for a detail referenced by the UPON phrase. If two or more instances of data-name-1 or identifier-1 specify the same data item, this data item is added into the sum counter as many times as data-name-1 or identifier-1 is referenced in the SUM clause. It is permissible for the UPON phrase to contain two or more instances of data-name-2 that specify the same detail. When a GENERATE statement for such a detail is executed, the adding takes place as many times as data-name-2 appears in the UPON phrase.

8. If the addend is data-name-1 and data-name-1 is a repeating item, repetitions of the addend are either all added into the same sum counter or are each added into a different corresponding occurrence of the sum counter, according to the following rules:

    a. If the addend and the sum counter are subject to the same number of levels of repetition, each occurrence of the addend is added into the corresponding occurrence of the sum counter.

    b. If the addend is subject to a greater number of levels of repetition than the sum counter, the number of levels of repetition of the addend is effectively reduced to that of the sum counter by forming the total of a complete table of occurrences of the addend at one or more levels into each occurrence of the sum counter. The levels at which these totals are formed are those of the highest level of repetition of the addend, excluding any OCCURS, multiple LINE, or multiple COLUMN clauses to which the addend and the entry containing the SUM clause are both subject.

9. If the SUM clause specifies more than one addend, the result is the same as when all the addends were summed separately according to the above rules and the results added together.
10. If the entry is associated with an absent data item as a result of a PRESENT WHEN clause or an OCCURS clause with the DEPENDING phrase, the corresponding sum counter is not printed and is not reset to zero for the current instance of the report group.
11. If the operand is data-name-1 and is declared to be absent as a result of a PRESENT WHEN clause or an OCCURS clause with the DEPENDING phrase, data-name-1 is not added into the sum counter during the processing instance of the report group in which data-name-1 is defined.
12. It is permissible for procedure division statements to alter the content of sum counters.

SYNCHRONIZED clause

The `SYNCHRONIZED` clause specifies the alignment of an elementary item on the natural boundaries of the computer memory (see Item Alignment).

General format

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

Syntax rules

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

General rules

1. The SYNCHRONIZED clause has effect only on numeric binary variables (COMP, COMP-4, COMP-5, BINARY) when at least one of the compiler options -cv, -ca, -dcmi is specified. Otherwise, the SYNCHRONIZED clause is treated as commentary only.
2. If either the -dcmi or the -cv compiler option is specified, then the SYNCHRONIZED clause on a non-elementary item applies also to its subordinate items, otherwise the SYNCHRONIZED clause affects only elementary items.
3. This clause specifies that the subject data item is to be aligned in the computer such that no other data item occupies any of the character positions between the leftmost and rightmost natural boundaries delimiting this data item.

   If the number of character positions required to store this data item is less than the number of character positions between those natural boundaries, the unused character positions (or portions thereof) must not be used for any other data item. Such unused character positions, however, are included in:

   a. The size of any group item(s) to which the elementary item belongs; and
   b. The number of character positions allocated when any such group item is the object of a REDEFINES clause. The unused character positions are not included in the character positions redefined when the elementary item is the object of a REDEFINES clause.
4. 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.
5. 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.
6. 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.
7. 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, such as justification, truncation, or overflow.
8. 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.
9. When the SYNCHRONIZED clause is specified in a data description entry of a data item that also contains an OCCURS clause, or in a data description entry of a data item subordinate to a data description entry that contains an OCCURS clause, then:

   a. Each occurrence of the data item is SYNCHRONIZED.
   b. Any implicit FILLER generated for other data items within that same table are generated for each occurrence of those data items.

TYPE clause

The TYPE clause in a report group description entry identifies the circumstances under which a report group will be printed.

General format

```
TYPE IS { REPORT HEADING }
        { RH }

        { PAGE HEADING }
        { PH }

        { CONTROL HEADING } [{ ON } ] { data-name-1 } [ OR PAGE ]
        { CH }              [{ FOR }] { FINAL }

        { DETAIL }
        { DE }

        { CONTROL FOOTING } [{ ON } ] { data-name-2 } [ OR PAGE ]
        { CF }              [{ FOR }] { FINAL }

        { PAGE FOOTING }
        { PF }

        { REPORT FOOTING }
        { RF }
```

Syntax rules

1. RH is an abbreviation for REPORT HEADING.
   PH is an abbreviation for PAGE HEADING.
   CH is an abbreviation for CONTROL HEADING.
   DE is an abbreviation for DETAIL.
   CF is an abbreviation for CONTROL FOOTING.
   PF is an abbreviation for PAGE FOOTING.
   RF is an abbreviation for REPORT FOOTING.
2. Data-name-1 and data-name-2 may be qualified and reference modified. If data-name-1 or data-name-2 is reference modified, leftmost-position and length shall be integer literals. Each data-name-1, data-name-2, and FINAL, if specified, shall be the same as one of the operands of the CONTROL clause of the corresponding report description entry.
3. Following CONTROL HEADING or CONTROL FOOTING, data-name-1, data-name-2, or FINAL may be omitted only if the CONTROL clause in the corresponding report description entry contains exactly one operand.
4. PAGE HEADING and PAGE FOOTING and the OR PAGE phrase are allowed only if a PAGE clause that defines the page limit is specified in the report description entry.
5. REPORT HEADING, PAGE HEADING, REPORT FOOTING, and PAGE FOOTING may each appear no more than once in any given report description.
6. At most one CONTROL HEADING and at most one CONTROL FOOTING may be defined for each control data item or FINAL of the CONTROL clause for any given report. Either or both of CONTROL HEADING and CONTROL FOOTING may be omitted for any given level of control.
7. Groups of type DETAIL, CONTROL HEADING, and CONTROL FOOTING are referred to as body groups. Each report description shall include at least one body group.
8. If no GENERATE data-name statements are specified in the procedure division, the report description need

not contain a DETAIL.

General rules

1. Report groups are printed only during the execution of a GENERATE or a TERMINATE statement. Detail report groups are printed when referenced explicitly in a GENERATE statement. All other report groups are printed implicitly according to the general rules that follow.
2. The conditions under which a given report group is printed depend on its type as follows:

    a. The report heading, if defined, is printed as the first report group in the report, when the chronologically first GENERATE statement for the report, if any, is executed.

    b. The page heading, if defined, is printed immediately before, and on the same page as, the chronologically first body group to be printed for the report, and subsequently as the first report group in each new page whenever a page advance takes place, except when the report group about to be printed is a report footing on a page by itself.

    c. Each control heading without the OR PAGE phrase, wherever defined, is printed automatically, in either of the following events:

        i. when the chronologically first GENERATE statement following an INITIATE statement for the report is executed, in order of control levels from highest to lowest,

        ii. immediately preceding any detail printed as the result of the execution of a GENERATE statement when a control break has been detected, in order of control levels from the level of the control break down to the lowest. The OR PAGE phrase causes the associated control heading to be printed in addition after each page advance, following any page heading, provided that the page advance did not take place just before the printing of a control footing at a lower control level.

    d. A detail is printed as the result of the execution of an explicit GENERATE statement that references it.

    e. Each control footing, wherever defined, is printed automatically in either of the following events:

        i. when a GENERATE statement is executed where a control break has been detected, preceding any control heading and detail groups, in order of controls from the lowest up to the level of the control break

        ii. when the TERMINATE statement is executed for the report, provided that at least one GENERATE statement has been executed after the chronologically last INITIATE statement for the report, in order of controls from lowest to highest.

    f. The page footing, if defined, is printed as the last report group on each page of the current report, except in the following cases:

        i. on the first page, if it is occupied only by a report heading group;

        ii. on the last page, if it is occupied only by a report footing group;

        If a report footing is defined and is not on a page by itself, the page footing on the last page is immediately followed by the report footing.

    g. The report footing, if defined, is printed, as the very last report group in the report, when a TERMINATE for the report is executed, provided that at least one GENERATE statement has been executed for the report since the chronologically last INITIATE statement was executed for the report.

3. The upper limit is defined to be the uppermost permitted line on the page that may be occupied by the report group's first line. It is calculated as follows:

    a. The upper limit for a report heading or a page heading where no report heading appears on the same page is the line given by the HEADING integer.

    b. The upper limit for a page heading where a report heading appears on the same page is the line following the last line of the report heading.

c. The upper limit for a body group, if there is no control heading defined in the report with the OR PAGE phrase, is the line given by the FIRST DETAIL integer.

d. If there is at least one control heading defined in the report with the OR PAGE phrase, the upper limit for a body group is determined as follows:

   i. If the body group is a control heading at the same or a higher control level than the highest-level control heading that has an OR PAGE phrase, the upper limit is the line given by the FIRST DETAIL integer.

   ii. If the body group is a control heading at a lower control level than the highest-level control heading that has an OR PAGE phrase, the upper limit is the line following the last line of the next higher-level control heading.

   iii. If the body group is a detail, the upper limit is the line following the last line of the lowest-level control heading that has an OR PAGE phrase.

   iv. If the body group is a control footing, the upper limit is the line following the last line of the lowest-level control heading with an OR PAGE phrase at the same level as the control footing, or higher. If no such control heading is defined, the upper limit is the FIRST DETAIL integer.

e. The upper limit for a page footing is the line number obtained by adding 1 to the FOOTING integer.

f. The upper limit for a report footing that appears on a page by itself is the line given by the HEADING integer.

g. The upper limit for a report footing that does not appear on a page by itself is the line following the last line of the page footing, if specified, or the line number obtained by adding 1 to the FOOTING integer, if no page footing is defined for the report.

4. The lower limit is defined to be the lowermost permitted line on the page that may be occupied by the report group's last line. It is calculated as follows:

   a. The lower limit for a report heading that appears on a page by itself is the page limit.

   b. The lower limit for a report heading that does not appear on a page by itself is the line preceding the first line of the page heading. If no page heading is defined for the report, the lower limit is the value obtained by subtracting 1 from the FIRST DETAIL integer.

   c. The lower limit for a page heading is the line number obtained by subtracting 1 from the FIRST DETAIL integer.

   d. The lower limit for a control heading is the line given by the LAST CONTROL HEADING integer.

   e. The lower limit for a detail is the line given by the LAST DETAIL integer.

   f. The lower limit for a control footing is the line given by the FOOTING integer.

   g. The lower limit for a page footing or report footing is the page limit.

GROUP-DYNAMIC clause

The GROUP-DYNAMIC clause specifies that a group item includes items with dynamic length. These sub items can be either ANY LENGTH items or dynamic capacity tables.

General Format

```
[ GROUP-DYNAMIC ]
```

Syntax Rules

1. The `GROUP-DYNAMIC` clause may be specified for group data items.

General Rules

1. Even if not explicitly specified, the `GROUP-DYNAMIC` clause is assumed for all the group items that include at least one sub item with dynamic length.
2. The dynamic length items are treated separately, as if they were not part of the group item.
3. When a group is moved to another group, the runtime moves the group stripped from the dynamic length items first. After it, it initializes the dynamic length items in the destination group and then it moves the dynamic length items of the source group following their ordinal position: the first dynamic length item in the source group is moved to the first dynamic length item in the destination group, the second dynamic length item in the source group is moved to the second dynamic length item in the destination group, and so on until the last dynamic length item in the source group has been moved. For a successful result the source group item and the destination group item should have the same structure.
4. When a group is compared with another group, the runtime compares the groups stripped from the dynamic length items first. After it, it compares the dynamic length items following their ordinal position: the first dynamic length item in the source group is compared with the first dynamic length item in the destination group, the second dynamic length item in the source group is compared with the second dynamic length item in the destination group, and so on until the last dynamic length item in the source group has been compared. For a successful comparison the source group item and the destination group item should have the same structure.

Example

The following source codes are equivalent. The second one demonstrates how the runtime internally manages the first one:

- a program that takes advantage of the GROUP-DYNAMIC feature

```
       program-id. grdyn-test.
       working-storage section.
       01 group-1 group-dynamic.
          03 item-1 pic x.
          03 item-2 occurs dynamic capacity cap1.
             05 item-2-sub pic x.
          03 item-3 pic x.
          03 item-4 pic x any length.

       01 group-2 group-dynamic.
          03 item-1-b pic x.
          03 item-2-b occurs dynamic capacity cap2.
             05 item-2-b-sub pic x.
          03 item-3-b pic x.
          03 item-4-b pic x any length.

       procedure division.
       main-logic.
           move "A" to item-1.
           move "B" to item-2-sub(1).
           move "C" to item-2-sub(2).
           move "D" to item-3.
           move "E" to item-4.

           move group-1 to group-2.

           if group-1 = group-2
              display "Data moved correctly [ok]"
           else
              display "Unexpected result [fail]"
           end-if.
           goback.
```

- a program that doesn't take advantage of the GROUP-DYNAMIC feature

```
program-id. grdyn-test.
working-storage section.
01 group-1.
   03 item-1 pic x.
   03 item-3 pic x.
01 item-2.
   03 filler occurs dynamic capacity cap1.
      05 item-2-sub pic x.
01 item-4 pic x any length.

01 group-2.
   03 item-1-b pic x.
   03 item-3-b pic x.
01 item-2-b.
   03 filler occurs dynamic capacity cap2.
      05 item-2-b-sub pic x.
01 item-4-b pic x any length.

77 i              pic 9(3).
77 flg-comp       pic 9 value 0.
   88 tables-equal value 1.
   88 tables-diff  value 0.

procedure division.
main-logic.
    move "A" to item-1.
    move "B" to item-2-sub(1).
    move "C" to item-2-sub(2).
    move "D" to item-3.
    move "E" to item-4.

    move group-1 to group-2.
    initialize item-2-b.
    perform varying i from 1 by 1 until i > cap1
       move item-2-sub(i) to item-2-b-sub(i)
    end-perform.
    move item-4 to item-4-b.


    set tables-equal to true.
    perform varying i from 1 by 1 until i > cap1
       if item-2-sub(i) not = item-2-b-sub(i)
          set tables-diff to true
          exit perform
       end-if
    end-perform.
    if group-1 = group-2 and
       tables-equal      and
       item-4 = item-4-b
       display "Data moved correctly [ok]"
    else
       display "Unexpected result [fail]"
    end-if.
    goback.
```

GROUP-USAGE clause

The GROUP-USAGE clause specifies the format of a group item in the computer storage.

General Format

```
[ GROUP-USAGE IS ] Usage-String
```

Usage-String

```
{NATIONAL         }
```

Syntax Rules

1. The GROUP-USAGE clause may be specified for group data items.
2. All subordinate items should be national or numeric-edited.

General Rules

1. The GROUP-USAGE IS NATIONAL clause specifies that a national data format is used to represent a data items in the storage of the computer.

USAGE clause

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

General format

```
[ USAGE IS ] {Usage-String-1}
             {Usage-String-2}
```

Usage-String-1 format:

```
  { DISPLAY          }
{ {BINARY           [(size)] } }
  {COMP-5           [(size)] }
  {COMPUTATIONAL-5  [(size)] }
{ {COMP                      } }
  {COMPUTATIONAL             }
  {COMP-4           [(size)] }
  {COMPUTATIONAL-4  [(size)] }
{ {COMP-1          } }
  {COMPUTATIONAL-1}
{ {COMP-2          } }
  {COMPUTATIONAL-2}
{ {COMP-3          } }
  {COMPUTATIONAL-3}
  {PACKED-DECIMAL  }
{ {COMP-6          } }
  {COMPUTATIONAL-6}
{ {COMP-9          } }
  {COMPUTATIONAL-9}
{ {COMP-N          } }
  {COMPUTATIONAL-N}
{ {COMP-X          } }
  {COMPUTATIONAL-X}
{ {COMP-0          } }
  {COMPUTATIONAL-0}
```

Usage-String-2 format:

```
{ CDATA                                                          }
{ DOUBLE                                                         }
{ FLOAT                                                          }
{ INDEX                                                          }
{ OBJECT REFERENCE [ { [ FACTORY OF ] Class-Name [ ONLY ] } ] }
                   [ { [ FACTORY OF ] ACTIVE-CLASS  ]     }   }
{ {SIGNED-INT}                                                   }
  {INTEGER   }
{ SHORT                                                          }
{ INT                                                            }
{ LONG                                                           }
{ SIGNED-INT                                                     }
{ SIGNED-LONG                                                    }
{ SIGNED-SHORT                                                   }
{ UNSIGNED-INT                                                   }
{ UNSIGNED-LONG                                                  }
{ UNSIGNED-SHORT                                                 }
{ {HANDLE } [ OF { Control                } ]                    }
  {POINTER}      { MENU                     }
                 { SUBWINDOW                }
                 { THREAD                   }
                 { VARIANT                  }
                 { LAYOUT-MANAGER           }
                 { control-class            }
                 { FONT [DEFAULT-FONT    ] }
                        [FIXED-FONT      ]
                        [LARGE-FONT      ]
                        [MEDIUM-FONT     ]
                        [SMALL-FONT      ]
                        [TRADITIONAL-FONT]
{ SQL TYPE IS { BLOB   }  [(Lob-Length)]                         }
              { CLOB   }
              { DBCLOB }
```

Syntax rules

1. The `USAGE` clause may be specified for both group and elementary data items.
2. If the `USAGE` clause is written in the data description entry for a group item, it may also be written in the data description entry for any subordinate elementary item or group item, but the same usage must be specified in both entries.
3. An elementary data item whose declaration contains, or an elementary data item subordinate to a group item whose declaration contains, a `USAGE` clause specifying *Usage-String-1* must be declared with a `PICTURE` character-string that describes a numeric item, i.e., a `PICTURE` character-string that contains only the symbols 'P', 'S', 'V', and '9' (see the PICTURE clause). Data items with `USAGE COMP-N`, `USAGE COMP-5` and `USAGE COMP-X` may also be declared `PICTURE` character-string containing only 'X' symbols.
4. The words `COMP`, `COMPUTATIONAL`, `COMP-4`, `COMPUTATIONAL-4` and `BINARY` are equivalent.
5. `size` is a numeric literal.
6. The words `COMP-1` and `COMPUTATIONAL-1` are equivalent.
7. The words `COMP-2` and `COMPUTATIONAL-2` are equivalent.
8. The words `COMP-3` and `COMPUTATIONAL-3` are equivalent.
9. The words `COMP-5` and `COMPUTATIONAL-5` are equivalent.

10. The words COMP-6 and COMPUTATIONAL-6 are equivalent.
11. The words COMP-9 and COMPUTATIONAL-9 are equivalent.
12. The words COMP-N and COMPUTATIONAL-N are equivalent.
13. The words COMP-X and COMPUTATIONAL-X are equivalent.
14. The words SIGNED-INT and INTEGER are equivalent.
15. The BLANK WHEN ZERO, JUSTIFIED, PICTURE, SYNCHRONIZED, and VALUE clauses must not be specified for data items whose usage is *Usage-String-2*.
16. *Class-Name* may be a Nonnumeric Literal referring to a primitive Java type (i.e. "int") an existing class (i.e. "java.lang.Integer") or a class defined in the REPOSITORY Paragraph in the Configuration Section of the ENVIRONMENT DIVISION. When referring to a primitive Java type or to an existing class, you can add the "..." suffix to define a variable number of occurrences of the class (varargs).
17. Control may be one of the supported controls. See the Controls Reference for the complete list.
18. The words COMP-0, COMPUTATIONAL-0 and SIGNED-SHORT are equivalent.
19. The ACTIVE-CLASS phrase may be specified only in a factory definition, an instance definition, or the linkage or working-storage section of a method definition.
20. *Log-Length* is a numeric literal optionally followed by K to specify Kilobytes, M to specify Megabytes or G to specify Gigabytes. K, M and G are case insensitive. If the literal is not followed by K, M or G, the length is expressed in Bytes.

General rules

1. If the USAGE clause is written at a group level, it applies to each elementary item in the group.
2. The USAGE clause specifies the manner in which a data item is represented in the storage of a computer. It does not affect the use of the data item, although the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referred to. The USAGE clause may affect the radix or type of character representation of the item.
3. If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, the usage is implicitly DISPLAY.
4. Decimal storage means that only half of a byte is used to store the value of a digit in binary format.
5. Binary storage means that one byte is used to store the value of one or more digits.
6. Big Endian and Little Endian refer to sequencing methods used to store bytes. Big Endian (big units first) means that bytes are stored from left to right, that is, from lowest to highest memory address. Little Endian (little units first) means that bytes are stored from right to left, that is, from highest to lowest memory address.
7. The USAGE IS DISPLAY clause (whether specified explicitly or implicitly) specifies that a standard data format is used to represent a data item in the storage of the computer, and that the data item is aligned on a character boundary. The effective size of data item depends on the PICTURE and SIGN clauses. Each digit occupies one byte. If the data item is signed, the sign is stored in the rightmost byte. The SIGN clause, and the Compiler Options may affect this behavior. The following table shows a mapping of digits to characters stored in USAGE DISPLAY items when specifying the different data storage compatibility compiler options:

Compiler Option

DIGIT	-Dca -Dcb -Dcd -Dcdm -Dcm -Dcmi -Dcr	-Dcm -Dcmi	-Dci -Dcii -Dcn -Dcv	-Dca -Dcd -Dcdm -Dci -Dcii -Dcn	-Dcb	-Dcr
	Positive	Negative	Positive	Negative	Negative	Negative
0	'0'	'p'	'{'	'}'	'@'	x'20'
1	'1'	'q'	'A'	'J'	'A'	!
2	'2'	'r'	'B'	'K'	'B'	"
3	'3'	's'	'C'	'L'	'C'	#
4	'4'	't'	'D'	'M'	'D'	$
5	'5'	'u'	'E'	'N'	'E'	%
6	'6'	'v'	'F'	'O'	'F'	&
7	'7'	'w'	'G'	'P'	'G'	'
8	'8'	'x'	'H'	'Q'	'H'	(
9	'9'	'y'	'I'	'R'	'I'	)

8. The `USAGE IS COMPUTATIONAL` clause specifies that the numeric item is stored in Big Endian binary format. The data item may be signed.

   The effective size of data item depends on the PICTURE clause and compile flags, according to the following table:

Number of digits	Bytes -Dca -Dcb -Dcmi -Dci -Dcii -Dcv	Bytes -Dcd -Dcm -Dcr	Bytes -Dcdm	Bytes -Dci + -D1 -Dcn
1-2	2	1	1	1
3-4	2	2	2	2
5-6	4	3	3	4
7	4	3	4	4
8-9	4	4	4	4
10-11	8	5	5	8

12	8	5	6	8
13-14	8	6	6	8
15-16	8	7	7	8
17-18	8	8	8	8
19	12	8	9	12
20-21	12	9	9	12
22-23	12	10	10	12
24	12	10	11	12
25-26	12	11	11	12
27-28	12	12	12	12
29-31	16	13	13	16

If `size` is specified, it takes priority on the above table.

9. The `USAGE IS COMPUTATIONAL-1` clause specifies that the numeric item is stored in Big Endian binary format. The data item occupies two bytes and it is always signed. It can store values from -32767 to +32767, regardless of the PICTURE clause. Under the -cv compiler option, COMPUTATIONAL-1 is treated as FLOAT and occupies four bytes.
10. The `USAGE IS COMPUTATIONAL-2` clause specifies that the numeric item is stored in Big Endian decimal format. Each digit occupies one byte and uses only the low-order half-byte. When the data item is signed, an additional trailing byte is used.
    The following table shows the rightmost byte value used to represent the sign representation for USAGE COMP-2 items when specifying the different data storage compatibility compiler options:

Compiler Option		
-Dca **-Dcd** **-Dcdm**	**-Dcb** **-Dci** **-Dcii** **-Dcm** **-Dcmi** **-Dcn** **-Dcr** **-Dcv**	**-Dca** **-Dcb** **-Dcd** **-Dcdm** **-Dci** **-Dcii** **-Dcm** **-Dcmi** **-Dcn** **-Dcr** **-Dcv**
Positive	**Positive**	**Negative**
x'0B'	x'0C'	x'0D'

Under the -cv compiler option, COMPUTATIONAL-2 is treated as DOUBLE.

11. The `USAGE IS COMPUTATIONAL-3` clause specifies that the numeric item is stored in Big Endian decimal format. Each digit occupies half of a byte. The last trailing half-byte is always reserved for sign, even if the

PICTURE clause does not contain the 'S' symbol.

The effective size of data item depends on the PICTURE clause, according to the following table:

Number of digits	Bytes
1	1
2-3	2
4-5	3
6-7	4
8-9	5
10-11	6
12-13	7
14-15	8
16-17	9
18-19	10
20-21	11
22-23	12
24-25	13
26-27	14
28-29	15
30-31	16

The following table shows the rightmost nibble value used to represent the sign representation for USAGE COMP-3 items when specifying the different data storage compatibility compiler options:

Compiler Option					
-Dca -Dcd -Dcdm	-Dcb -Dci -Dcii -Dcm -Dcmi -Dcr -Dcv	-Dcn	-Dca -Dcb -Dcd -Dcdm -Dci -Dcii -Dcm -Dcmi -Dcn -Dcr -Dcv	-Dca -Dcb -Dcd -Dcdm -Dci -Dcii -Dcm -Dcmi -Dcn -Dcr	-Dcv
Positive	Positive	Positive	Negative	Unsigned	Unsigned

x'0F'	x'0C'	x'0B'	x'0D'	x'0F'	x'0C'

12. The `USAGE IS COMPUTATIONAL-5` is primarily used to communicate with external programs that expect native data storage. The format of a `COMPUTATIONAL-5` data item is identical to a `COMPUTATIONAL` data item, except that the data is stored in a machine-dependent format. A `VALUE` clause for a `COMPUTATIONAL-5` data item is stored in a machine-independent format and is adjusted when it is loaded into the data item. If `COMPUTATIONAL-5` is used with a `PIC X(n)` data item and assigned an alphanumeric value, the results are undefined. A `PIC X(n)` data item used with `COMPUTATIONAL-5` cannot be signed.
13. The `USAGE IS COMPUTATIONAL-6` clause specifies that the numeric item is stored in Big Endian decimal format. Each digit occupies half of a byte. The data item may not be signed.

    The effective size of data item depends on the PICTURE clause, according to the following table:

Number of digits	Bytes
1-2	1
3-4	2
5-6	3
7-8	4
9-10	5
11-12	6
13-14	7
15-16	8
17-18	9
19-20	10
21-22	11
23-24	12
25-26	13
27-28	14
29-30	15
31	16

14. The `USAGE IS COMPUTATIONAL-9` clause specifies that the numeric item is stored in Big Endian decimal format. Each digit occupies half of abyte. The first leading half-byte is always reserved for sign, even if the PICTURE clause does not contain the 'S' symbol.

    The effective size of data item depends on the PICTURE clause, according to the following table:

Number of digits	Bytes

1	1
2-3	2
4-5	3
6-7	4
8-9	5
10-11	6
12-13	7
14-15	8
16-17	9
18-19	10
20-21	11
22-23	12
24-25	13
26-27	14
28-29	15
30-31	16

15. The `USAGE IS COMPUTATIONAL-N` clause specifies that the numeric item is stored in Big Endian binary format.

    The effective size of data item depends on the PICTURE clause, according to the following table:

Number of digits	Bytes
1-2	1
3-4	2
5-7	3
8-9	4
10-12	5
13-14	6
15-16	7
17-18	8

When the PICTURE clause contains only 'X' symbols, the number of 'X's represent the actual storage size of the data item. The maximum size of the item is 8. The maximum value that the data item can store is summarized in the following table:

Size	Value
1	255
2	65.535
3	16.777.215
4	4.294.967.295
5	1.099.511.627.775
6	281.474.976.710.655
7	72.057.594.037.927.935
8	9.223.372.036.854.775.807

16. The `USAGE IS COMPUTATIONAL-X` clause specifies that the numeric item is stored in Big Endian binary format.

    The effective size of data item depends on the PICTURE clause, according to the following table:

Number of digits	Bytes
1-2	1
3-4	2
5-7	3
8-9	4
10-12	5
13-14	6
15-16	7
17-19	8
20-21	9
22-24	10
25-26	11
27-28	12
29-31	13

When the PICTURE clause contains only 'X' symbols, the number of 'X's represent the actual storage size of the data item. The maximum allowed size of the item is 8. The maximum value that the data item can store is summarized in the following table:

Size	Value
1	255
2	65.535
3	16.777.215
4	4.294.967.295
5	1.099.511.627.775
6	281.474.976.710.655
7	72.057.594.037.927.935
8	9.223.372.036.854.775.807

17. The `USAGE IS DOUBLE` clause specifies that the numeric item is stored as a floating point double precision type.
18. The `USAGE IS FLOAT` clause specifies that the numeric item is stored as a floating point single precision type.
19. The `USAGE IS INDEX` clause specifies that the numeric item is stored in Big Endian binary format. The data item occupies four bytes. The data item can store positive integers ranging from 0 to 4.294.967.295.
20. The `USAGE IS OBJECT` clause specifies that the data item is a reference to an object.

    a. The `USAGE IS OBJECT` clause may be specified only for 01 or 77 data items.

    b. If none of the optional phrases is specified, the data item being defined is called a universal object reference. Its content may be a reference to any object.

    c. If *Class-Name* is specified, the object referenced by this data item shall be an object of *Class-Name* or of a subclass of *Class-Name*, subject to the following rules:

        i. If the `FACTORY` phrase is specified, the object referenced by this data item shall be the factory object of the specified class.

        ii. If the `FACTORY` phrase is not specified, the object referenced by this data item shall be an instance object of the specified class.

    The `ONLY` phrase is treated as a commentary.

    When *Class-Name* is suffixed with "..." the compiler internally treats it as array of objects of *Class-Name*.
    If a C*lass-Name* is suffixed with "..." is used in the USING clause of PROCEDURE DIVISION or ENTRY POINT, it must be the last parameter and it's considered as varargs, so the runtime expects variable number of occurrences of this parameter.
    If a C*lass-Name* is suffixed with "..." is used in the WORKING-STORAGE SECTION, the compiler treats it as a standard array. In this case "...". is synonymous of "[]".

d. If ACTIVE-CLASS is specified, the object referenced by this data item shall be of the same class as the object that was used to invoke the method in which this data description entry is specified, subject to the following rules

   i. If the FACTORY phrase is specified, the object referenced by this data item shall be the factory object of the specified class.

   ii. If the FACTORY phrase is not specified, the object referenced by this data item shall be an instance object of the specified class.

21. The USAGE IS SIGNED-INT clause specifies that the numeric item is stored as a C-style int type.
22. The USAGE IS SIGNED-LONG clause specifies that the numeric item is stored as a C-style long type.
23. The USAGE IS SIGNED-SHORT clause specifies that the numeric item is stored as a C-style short type.
24. The USAGE IS UNSIGNED-INT clause specifies that the numeric item is stored as a C-style unsigned int type.
25. The USAGE IS UNSIGNED-LONG clause specifies that the numeric item is stored as a C-style unsigned long type.
26. The USAGE IS UNSIGNED-SHORT clause specifies that the numeric item is stored as a C-style unsigned short type.
27. The USAGE IS HANDLE clause specifies that the data item refers to a COBOL object.

    a. If none of the optional phrases is specified, the data item being defined is a generic handle. Its type is returned by the HANDLE-TYPE Function.

    b. Items defined as USAGE IS HANDLE OF Control can be used with the DISPLAY, ACCEPT, MODIFY, INQUIRE and DESTROY Statements.

    c. Items defined as USAGE IS HANDLE OF MENU can be used with the "W$MENU" Routine and as an argument of the POP-UP Property of Controls.

    d. Items defined as USAGE IS HANDLE OF SUBWINDOW can be used with the DISPLAY and CLOSE Statements.

    e. Items defined as USAGE IS HANDLE OF THREAD can be used with the ACCEPT, CALL, PERFORM, RECEIVE, SEND and WAIT Statements.

    f. Items defined as USAGE IS HANDLE OF FONT can be used with the "W$FONT" Routine, as an argument of the FONT and CONTROL FONT Properties of Controls and with the DESTROY Statement.

       When one of the optional phrases DEFAULT-FONT, FIXED-FONT, LARGE-FONT, MEDIUM-FONT, SMALL-FONT or TRADITIONAL-FONT is specified, the data item being defined will refer to the corresponding font. The iscobol.font.* properties can be used to set the fonts to be used.

28. USAGE POINTER requires either -ca or -cp compiler options to be compiled correctly. In the first case, it's treated as USAGE HANDLE, in the second case it's treated as a real POINTER and can be shared with external C routines.
29. USAGE VARIANT is reserved for future use and it's currently treated as USAGE DISPLAY.
30. USAGE CDATA can be used only for variable whose parent is an XML element, otherwise an error is raised. When writing an XML document the content of variable with such a USAGE is generated between the CDATA tags. For example, the following variable:

```
05 xml-cdata pic x any length cdata value "<<<TEST>>>".
```

generates:

```
<![CDATA[<<<TEST>>>]]>
```

If a variable contains the sequence "]]>" in some point, then as many CDATA tags as the number of such occurrences are generated in order to be compliant with the XML syntax.
This clause doesn't work in conjunction with BASE64BINARY, BOOLEAN, HEXBINARY and RAW caluses.

31. The `USAGE IS COMPUTATIONAL-0` is a Microsoft COBOL extension and requires the -cms compile flag.
32. The `USAGE IS SQL TYPE` is used in programs with Embedded SQL to specify that the host variable maps a large object field on the table. The data item is internally expanded in a group item where data and length are held in separate fields. For example, the following data definition:

```
01 BLOB-ITEM USAGE IS SQL TYPE IS BLOB(2M).
```

is expanded by the compiler to

```
01 BLOB-ITEM.
   49 BLOB-ITEM-LENGTH PIC S9(9) COMP-5.
   49 BLOB-ITEM-DATA PIC X(2097152).
```

VALUE clause

The `VALUE` clause defines the initial value of data items.

Format 1

```
VALUE IS {Literal-1} [ {+} Literal-2  [ ... {+} Literal-3] ] .
         {NEXT     }   {-}                 {-}
                       {*}                 {*}
                       {/}                 {/}
                       {**}                {**}
```

Format 2

```
   {VALUE IS  } { Literal-1 [ {THROUGH} Literal-2 ] } ...
   {VALUES ARE}             {THRU   }

       [ WHEN SET TO FALSE Literal-3 ]
```

Syntax rules

1. *Literal-1*, *Literal-2* and *Literal-3* are Literal, as defined in the Definitions section in the Preface of this document. The category of Literal-1 must match the category of the item being described.
2. A signed Numeric Literal must have a signed numeric PICTURE character-string associated with it.
3. All Numeric Literals in a `VALUE` clause of an item must have a value which is within the range of values indicated by the PICTURE clause, and must not have a value which would require truncation of nonzero digits. Nonnumeric Literals in a `VALUE` clause of an item must not exceed the size indicated by the PICTURE clause.
4. The `VALUE` clause must not be specified in any entry which is part of the description or redefinition of an external data record.

General rules

Format 1

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

    a. If the category of the item is numeric, all literals in the VALUE clause must be numeric. If the literal defines the value of a working storage item, the literal is aligned in the data item according to the Standard Alignment Rules.

    b. If the category of the item is alphabetic, alphanumeric, alphanumeric edited, or numeric edited, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric (see Standard Alignment Rules). 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 (see The PICTURE clause). Therefore, the VALUE for an edited item must be specified in an edited form.

    c. Initialization is not affected by any BLANK WHEN ZERO or JUSTIFIED clause that may be specified.

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

    a. The VALUE clause cannot be used in the FILE Section. In the FILE Section, the VALUE clause may be used only in condition-name entries; therefore, the initial value of the data items in the FILE Section is undefined.

    b. The VALUE clause cannot be used in the LINKAGE Section. In the LINKAGE Section, the VALUE clause may be used only in condition-name entries.

    c. VALUE clauses in the WORKING-STORAGE Section of a program take effect only when the program is placed into its initial state. If the VALUE clause is used in the description of the data item, the data item is initialized to the defined value. If the VALUE clause is not associated with a data item, the initial value of that data item is low-values if the EXTERNAL clause is specified, spaces otherwise.

3. The VALUE clause must not be stated in a data description entry that contains a REDEFINES clause, or in an entry that is subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

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

5. The VALUE clause must not be specified for a group item containing items subordinate to it with descriptions including JUSTIFIED, SYNCHRONIZED or USAGE (other than USAGE IS DISPLAY).

6. If a VALUE clause is specified in a data description entry of a data item which is associated with a variable occurrence data item, the initialization of the data item behaves as if the value of the data item 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 which 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.

7. 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.

8. A VALUE clause specified in a data description entry that contains an OCCURS clause or in a entry that is subordinate to an OCCURS clause causes every occurrence of the associated data item to be assigned the specified value.

9. The value returned by NEXT is the offset at which the next byte of storage occurs after the previous data declaration. (It is not the offset of the start of the next data declaration.) If that data declaration was of a table defined with an OCCURS clause, the value returned by NEXT is the offset at which the next byte of storage occurs after the first element of the table. If the identifier is part of an EXTERNAL record or a LINKAGE record, the offset is calculated from the start of the associated 01-level, otherwise the offset is calculated from the start of the Data Division. If identifier is a constant, the NEXT phrase points to the offset at which the next byte of storage occurs after the previous data declaration.
10. The `NEXT` clause requires -m1 compiler option in order to work correctly.
11. When `VALUE` is followed by an arithmetic operation:

    a. The result of the operation is always an integer, if necessary a scale will be applied

    b. Arithmetic operations are executed sequentially in the order they appear in the VALUE clause and not by respecting algebraic precedence rules.
12. When a `VALUE` clause is applied to an edited item, that item is treated as if it were alphanumeric. Editing characters in the `PICTURE` clause count toward the size of the item but have no effect on initialization.

Format 2

    c. A Format 2 `VALUE` clause defines a Condition-name condition.

VARYING clause

The VARYING clause establishes counters to enable different source items to be placed in each occurrence of a repeated printable item in report writer.

The VARYING clause also establishes counters to enable a VALIDATE statement to store different occurrences of a data item in different occurrences of a destination data item, or to compare different occurrences of a data item during content or relation validation.

General format

```
VARYING { data-name-1 [ FROM arithmetic-expression-1 ] [ BY arithmetic-expression-2 ] }
    ...
```

Syntax rules

1. The entry containing the VARYING clause shall also contain an OCCURS clause or, if the VARYING clause appears in a report group description entry, a multiple LINE or multiple COLUMN clause.
2. Data-name-1 shall not be defined elsewhere in the source element, except as data-name-1 in another VARYING clause of an entry not subordinate to the subject of the current entry.

   NOTE: Such a reuse refers to a completely independent data item.

   This definition of data-name-1 may be referenced only within the current entry or a subordinate entry.
3. Data-name-1 shall not be referenced in arithmetic-expression-1 of the same VARYING clause, but may be referenced in arithmetic-expression-2 of the same VARYING clause or in arithmetic-expression-1 or arithmetic-expression-2 of a VARYING clause in a subordinate entry.

General rules

1. Each entry containing a VARYING clause establishes an independent temporary integer data item that shall be large enough to contain the maximum expected value.
2. If the VARYING clause is not specified in a report description entry, it is ignored during the execution of any statement other than a VALIDATE statement.

3. When the data item that is the subject of the entry is processed by the VALIDATE statement or the report item that is the subject of the entry is processed, the content of data-name-1 is established for each repetition of the associated data item during each stage of execution of the VALIDATE statement or each repetition of the report item, as follows:

    a. For the first occurrence, the value of arithmetic-expression-1 is moved to data-name-1. If the FROM phrase is absent, 1 is moved to data-name-1.

    b. For the second and subsequent occurrences, the value of arithmetic-expression-2 is added to data-name-1. If the BY phrase is absent, 1 is added to data-name-1.

4. Each value of data-name-1, established in this way, persists throughout the processing of the associated occurrence of the data item or report item.

    NOTE: For example, this allows data-name-1 to be used as a source data item, as a subscript to a source data item or as part of the identifier in the DEFAULT clause.

5. If the evaluation of arithmetic-expression-1 or arithmetic-expression-2 produces a non-integer value and the VARYING clause was specified in a report description entry, the EC-REPORT-VARYING exception condition is set to exist, the execution of the GENERATE statement is unsuccessful, and the content of the print line is undefined.

6. If the evaluation of arithmetic-expression-1 or arithmetic-expression-2 produces a non-integer value and the VARYING clause was not specified in a report description entry, the EC-VALIDATE-VARYING exception condition is set to exist, the execution of the VALIDATE statement is unsuccessful, and the content of the receiving items is undefined.

SAME AS clause

The SAME AS clause causes a data item to inherit the same definition of another data item.

General format

```
SAME AS data-name-1.
```

Syntax rules

1. data-name-1 is a Data Item defined elsewhere in the same program.

Screen Description

A screen description entry specifies attributes, behavior, size, and location of a screen item so that it can be referenced by an ACCEPT screen or a DISPLAY screen Statement. The screen description entry allows data items to be associated with the screen item so that the contents of the data item are displayed within the screen item or the value keyed into a screen item by the operator is placed in the data item.

General Format

```
Level-Number { Screen-Name } [ AUTO clause            ]
             { FILLER      } [ Background Intensity   ]
                             [ Foreground Intensity   ]
                             [ {BELL}                 ]
                               {BEEP}
                             [ BLANK clause           ]
                             [ BLANK WHEN ZERO clause ]
                             [ {BLINKING}             ]
                               {BLINK   }
                             [ COLOR clause           ]
                             [ COLUMN clause          ]
                             [ CONTROL clause         ]
                             [ LINE clause            ]
                             [ FROM clause            ]
                             [ {FULL        }         ]
                               {LENGTH-CHECK}
                             [ JUSTIFIED clause       ]
                             [ NO-ECHO clause         ]
                             [ OCCURS clause          ]
                             [ OUTPUT clause          ]
                             [ PICTURE clause         ]
                             [ PROMPT clause          ]
                             [ {REQUIRED   }          ]
                               {EMPTY-CHECK}
                             [ REVERSE clause         ]
                             [ SAME                   ]
                             [ SIGN clause            ]
                             [ SIZE clause            ]
                             [ TAB-GROUP clause       ]
                             [ TAB-GROUP-VALUE clause ]
                             [ TO clause              ]
                             [ UNDERLINED clause      ]
                             [ UPPER and LOWER clauses]
                             [ USING clause           ]
                             [ VALUE clause           ]
                             [ ZERO-FILL clause       ]
                             [ Embedded Procedures    ]
```

Syntax rules

1. *Screen-Name* is a User-defined word, as defined in the Definitions section in the Preface of this document.

General rules

1. *Level-Number* may be any number from 1 through 49.
2. Clauses may be specified in any order.
3. For an elementary screen item, the associated screen description entry shall include at least one of the following:

- a FROM, TO, USING or VALUE clause;
- a BLANK (or ERASE) clause;
- a BELL clause.

4. If the FULL clause is specified, the JUSTIFIED clause shall not be specified.
5. If the same clause, other than an OCCURS clause, is specified at more than one level in the hierarchy of a screen item, the clause that appears at the lowest level of the hierarchy is the one that takes effect.
6. If the HIGHLIGHT and LOWLIGHT clauses are both specified in the hierarchy of a screen item, the clause that appears at the lowest level of the hierarchy is the one that takes effect.
7. The PICTURE clause may be omitted when an alphanumeric, national or numeric literal is specified in the USING clause. A PICTURE clause is implied as follows:
   a. if the literal is alphanumeric, 'PICTURE X(*length*)'
   b. if the literal is national, 'PICTURE N(*length*)'
   c. if the literal is numeric, 'PICTURE 9(*length*)'
8. If the AUTO and TAB clauses are both specified in the hierarchy of a screen item, the AUTO clause takes effect. If the AUTO clause is implied (see the '-va' compile-time option in "Compiler Options" in the isCOBOL Evolve User's Guide) and the TAB clause is specified, the TAB clause takes effect.
9. BELL and BEEP are synonymous and are treated as a commentary.
10. BLINKING and BLINK are synonymous and are treated as a commentary.
11. FULL and LENGTH-CHECK are synonymous and are treated as a commentary.
12. REQUIRED and EMPTY-CHECK are synonymous and are treated as a commentary.
13. The SAME clause is treated as a commentary.

Level-Number

Level numbers 1 through 49 indicate the position of a data item or screen item within the hierarchical structure described by a screen description entry.

General format

```
Level-Number
```

General rules

1. The level-number 1 identifies the first entry in each screen description.

Syntax Rules

1. A level-number is required as the first element in each screen description entry.
2. A level-number in the range of 1 through 9 may be specified as 01 through 09.
3. Screen description entries shall have level-numbers 1 through 49.

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

```
{AUTO          }
{AUTO-SKIP     }
{AUTOTERMINATE}
{TAB           }
```

General rules

1. AUTO, AUTO-SKIP, AUTOTERMINATE and TAB are synonyms.
2. An AUTO clause specified at the group level applies to each input screen item in that group.
3. The AUTO clause is ignored for a field that is not an input field.
4. 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.
5. The AUTO clause causes the cursor to be automatically moved to the next input field declared for the screen item when the last character of the input field whose definition contains this clause has data entered into it.
6. 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 Intensity

The background intensity of a screen item is specified as follows.

General Format

```
 [ {BACKGROUND-HIGH    } ]
   {BACKGROUND-LOW     }
   {BACKGROUND-STANDARD}
```

BLANK clause

The BLANK clause clears a screen line or clears the whole screen during the execution of a DISPLAY screen Statement before data is transferred to the screen item.

General format

```
{BLANK} {SCREEN}
{ERASE} {LINE  }
        {EOS   }
        {EOL   }
```

General rules

1. BLANK and ERASE are synonyms.
2. When the BLANK SCREEN clause is specified, the screen is cleared and the cursor is placed at line 1, column 1 during the execution of a DISPLAY screen Statement before data is transferred to the screen item. Upon clearing the screen, the background color for the entire screen is set to the value applicable at the time.
3. 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 during the execution of a DISPLAY screen Statement before data is transferred to the screen item.

4. When the BLANK EOS clause is specified, the screen is cleared starting from the position of the screen item that is the subject of the entry through the end of the screen, during the execution of a DISPLAY screen Statement before data is transferred to the screen item.
5. When the BLANK EOL clause is specified, the line, from the position of the screen item that is the subject of the entry through the end of the line, is cleared during the execution of a DISPLAY screen Statement before data is transferred to the screen item.
6. 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.
7. 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.
8. The BLANK clause is ignored during all phases of the execution of an ACCEPT screen Statement.

BLANK WHEN ZERO clause

The BLANK WHEN ZERO clause causes the blanking of an item when a value of zero is being stored in it.

General format

```
BLANK WHEN ZERO
```

Syntax rules

1. The BLANK WHEN ZERO clause may be specified only for an elementary item described by its picture Character-String as category numeric-edited or as numeric without the picture symbol 'S'.
2. The subject of the entry shall be implicitly or explicitly described as usage display or usage national.

General rules

1. When the BLANK WHEN ZERO clause is specified for a data item, the content of the data item is set to all spaces when the item is a receiving operand and the value being stored is zero.
2. If the subject of the entry is described by its picture character-string as category numeric, the BLANK WHEN ZERO clause defines the item as numeric-edited.

COLOR clause

The COLOR clause causes the data to be shown with a particular color.

Format 1

```
COLOR IS {Data-Name-1}
         {Numeric-Literal-1}
```

Format 2

```
BACKGROUND-COLOR IS [RGB] {Data-Name-2}
                          {Numeric-Literal-2}
FOREGROUND-COLOR IS [RGB] {Data-Name-2}
                          {Numeric-Literal-2}
```

Format 3

```
BACKGROUND IS Color-Name-1
FOREGROUND IS Color-Name-1
```

Syntax rules

1. *Data-Name-1* and *Data-Name-2* are User-defined words, as defined in the Definitions section in the Preface of this document.
2. *Numeric-Literal-1* and *Numeric-Literal-2* are Numeric Literals, as defined in the Definitions section in the Preface of this document.
3. *Color-Name-1* may be any properly formed user-defined word that names a color known to the runtime system. The default names known to the runtime system are: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN and WHITE.

General Rules

Formats 1 and 2

1. See Color management for more information.

Format 3

1. The BACKGROUND clause causes the background of the screen field to be shown in the specified color.
2. The FOREGROUND clause causes the foreground of the screen field to be shown in the specified color.

COLUMN clause

The COLUMN clause identifies a printable item, or a set of printable items, and specifies their horizontal location in a report line, or specifies the horizontal screen coordinate for a screen item.

Format 1 (Report)

```
{ COLUMN   } { NUMBER  } { IS  } integer-1 { PLUS } integer-2
{ COLUMNS  } { NUMBERS } { ARE }           {   +  }
{ COL      }
{ COLS     }

      { LEFT    }
      { CENTER  }
      { RIGHT   }
```

Syntax rules

1. COLUMN, COL, COLUMNS, and COLS are synonyms.
2. PLUS and + are synonyms.
3. The COLUMN clause may be specified only in an elementary entry. The entry shall also contain a LINE clause or shall be subordinate to an entry containing a LINE clause.
4. The keyword ARE may be specified only if COLUMNS, COLS, or NUMBERS is specified.
5. The keyword IS shall not be specified if COLUMNS, COLS, or NUMBERS is specified.
6. Neither integer-1 nor integer-2 shall exceed the page width.
7. Within a given report line, any two or more absolute items defined using column numbers that are not in increasing numerical order shall be subject to a different PRESENT WHEN clause.
8. The set of printable items within a given report line is subject to the following rules:
    a. If any two or more items overlap each other, they shall each be subject to a different PRESENT WHEN clause.
    b. The rightmost column positions of all absolute items shall not exceed the page width
9. If the report line ends in a set of relative printable items or consists only of such, they shall not cause the page width to be exceeded unless each of them is subject to a different PRESENT WHEN clause, in which case this rule applies only to the largest of them.
10. If LEFT, CENTER, or RIGHT is specified, all the operands shall be absolute. If any of the operands is absolute and neither LEFT, CENTER, nor RIGHT is specified, LEFT is assumed.
11. If more than one integer-1 or integer-2 operand is specified the clause is referred to as a multiple COLUMN clause and the following additional rules apply:
    a. No OCCURS clause shall be specified in the same entry.
    b. All the occurrences of integer-1 shall be in increasing order of magnitude.

General rules

1. The COLUMN clause defines one or more printable items. If a COLUMN clause is not specified, the items are not printed.
2. There is a fixed correspondence, specified by the implementor, between a column and a character in a national character set.
3. The printable-size of a printable item is the number of columns required for printing the characters described by the item's PICTURE clause or, in the absence of a PICTURE clause, the literal specified in the VALUE clause. There is a one-to-one correspondence between a column and a character in an alphanumeric character set.

    NOTE: Columns might not line up in a report if the printable characters are not of a fixed size, such as in the characterset UTF-8.

4. Any report line shall be defined in such a way that any given column position is used for only one printable item when the line is printed. If this rule is violated the EC-REPORT-COLUMN-OVERLAP exception condition is set to exist and the results are undefined.
5. Any report line shall be defined in such a way that, when printed, the final column position of the last printable item does not exceed the page width. If this rule is violated the EC-REPORT-PAGE-WIDTH exception condition is set to exist, the report line is truncated, and the report line is printed.
6. If the integer-1 phrase is specified, the following general rules apply:

    a. Integer-1 specifies an absolute column number.

    b. If LEFT is specified, integer-1 is the leftmost column of the printable item. The rightmost column of the printable item is integer-1 + printable-size – 1.

    c. If RIGHT is specified, integer-1 is the rightmost column of the printable item. The leftmost column of the printable item is integer-1 – printable-size + 1.

    d. If CENTER is specified, the printable item is centered, as follows:

        i. If the printable-size of the printable item is an odd number of columns, integer-1 identifies the center column of the printable item. The leftmost column is integer-1 - ((printable-size - 1) / 2); the rightmost column is integer-1 + (printable-size / 2), truncated to an integer.

        ii. If the printable-size of the printable item is an even number of columns, integer-1 identifies the column of the printable item that is to the left of an imaginary line between the two middle columns of the printable item. The leftmost column is (integer-1 - (printable-size / 2)) + 1; the rightmost column is integer-1 + (printable-size / 2).
7. Within any given report line, a horizontal counter, representing the rightmost occupied column, is maintained and updated by each printable item in the line. At the start of the line, the horizontal counter is zero.
8. Integer-2 specifies a relative column number. If integer-2 is specified for an item that is the first printable item in the current line, it specifies the leftmost column of that item. Otherwise, the value of integer-2 is the number of column positions between the rightmost column of the preceding printable item and the leftmost column of the item being defined. The position of the item's leftmost character is obtained by adding integer-2 to the current line's horizontal counter.

    NOTE: The value of integer-2 minus 1 is the number of blank columns, if any, immediately preceding the item.
9. The rightmost column position of each printable item becomes the new value of the horizontal counter.
10. Any unoccupied columns in each print line are filled with space characters.
11. If an entry containing a LINE clause has no subordinate entry defining a printable item, the resultant report line will be blank.
12. A multiple COLUMN clause is functionally equivalent to a COLUMN clause with a single operand, together with a simple OCCURS clause whose integer is equal to the number of operands of the COLUMN clause, except that the multiple COLUMN clause allows the printable items to be defined at unequal horizontal intervals.

Format 2 (Screen)

```
{COLUMN  } [ NUMBER IS [PLUS] {Data-Name-3} ]
{COL     }              [+   ] {Integer-3  }
{POSITION}              [-   ]
{POS     }
```

Syntax rules

1. COLUMN, COL, POS and POSITION are synonyms.
2. PLUS and '+' are synonyms.
3. *Data-Name-3* shall be described in the file, working-storage, local-storage or linkage section as an elementary unsigned integer data item.
4. Neither the PLUS phrase nor the '-' phrase shall be specified for the first elementary item in a screen record.

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 or MINUS, 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 or '-' 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 *Data-Name-3* or *Integer-3*. '-' denotes a column position that is decreased by the value of *Data-Name-3* or *Integer-3*.
4. A setting of COLUMN 1 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 results are as if 1 were specified for the column number.

CONTROL clause

```
CONTROL [IS] Cntrl-String
```

General Rules

1. The CONTROL phrase provides the ability to modify the static attributes of the DISPLAY statement at runtime. The CONTROL data item is treated as a series of comma-separated keywords that control the action of the

statement. Within the CONTROL data item, spaces are ignored and lower-case letters are treated as if they were upper-case. The keywords allowed in cntrl-string are:

```
ERASE, ERASE EOL, ERASE EOS, NO ERASE
BEEP, NO BEEP
HIGH, HIGHLIGHT, LOW, STANDARD, OFF
BLINK, NO BLINK
REVERSE, NO REVERSE
TAB, NO TAB
PROMPT, NO PROMPT
CONVERT, NO CONVERT
UPDATE, NO UPDATE
ECHO, NO ECHO
UPPER, NO UPPER, LOWER, NO LOWER
UNDERLINED, NO UNDERLINE
LEFT, RIGHT, CENTERED, NO JUST
SAME
FCOLOR
BCOLOR
```

Any other keywords and spaces are discarded. If more than one keyword from within the above lines appears in cntrl-string, then only the rightmost one in the data item is used. Each of the keywords performs the same action as the statically declared attribute of the same name. When a CONTROL item conflicts with the statically declared attributes of the DISPLAY statement, the actions specified in the CONTROL item take precedence. The FCOLOR and BCOLOR keywords are used to set foreground and background colors respectively. These keywords must be followed by an equals sign and the name of a color taken from the following list: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, and WHITE. The named color becomes the default foreground or background color for the window. Note that this is different from the COLOR phrase, which sets the color only for the current DISPLAY statement. The FCOLOR and BCOLOR keywords set the default colors for every subsequent DISPLAY until explicitly changed.

Foreground Intensity

The foreground intensity of a screen item is specified as follows.

General Format

```
{HIGHLIGHT}
{HIGH     }
{BOLD     }
{LOWLIGHT }
{LOW      }
{STANDARD }
```

FROM clause

The FROM clause specifies the source of data for an ACCEPT screen Statement and a DISPLAY screen Statement.

General format

```
FROM {Data-Name-5}
     {Literal-1  }
```

Syntax rules

1. *Data-Name-5* shall be defined in the file, working-storage, local-storage or linkage section.
2. The category of *Data-Name-5* and *Literal-1* shall be a permissible category as a sending operand in a MOVE Statement where the receiving operand has the same PICTURE clause as the subject of the entry.
3. If the subject of this entry is subject to an OCCURS clause, *Data-Name-5* shall be specified without the subscripting normally required. Additional requirements are specified in the OCCURS clause.
4. *Data-Name-5* shall not specify a variable-length group.

General rules

1. The subject of the entry is an output screen item.

JUSTIFIED clause

The JUSTIFIED clause specifies right justification of data within a receiving data item or screen item.

General format

```
{JUSTIFIED} RIGHT
{JUST     }
```

Syntax rules

1. JUSTIFIED and JUST are synonyms.
2. The JUSTIFIED clause may be specified only at the elementary item level.
3. The JUSTIFIED clause may be specified only for a data item whose category is alphabetic, alphanumeric, or national.
4. The JUSTIFIED clause shall not be specified for an any-length elementary item.

General rules

1. When the receiving data item is described with the JUSTIFIED clause and the sending operand is larger than the receiving data item, the leftmost character positions of the sending operand shall be truncated.
2. When the receiving data item is described with the JUSTIFIED clause and it is larger than the sending operand, the data is aligned at the rightmost character position in the data item 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.
3. When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply. See Standard Alignment Rules.

LINE clause

The LINE clause specifies vertical positioning for its screen item.

General format

```
LINE [ NUMBER IS [PLUS] {Data-Name-6} ]
                [+   ] {Integer-5  }
                [-   ]
```

Syntax rules

1. PLUS and '+' are synonyms.
2. *Data-Name-6* shall be described in the file, working-storage, local-storage or linkage section as an elementary unsigned integer data item.
3. Neither the PLUS phrase nor the '-' phrase shall be specified for the first elementary item in a screen record.

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 or '-', 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 or '-' 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 *Data-Name-6* or *Integer-5*. '-' denotes a line position that is decreased by the value of D*ata-Name-6* or *Integer-5*.
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

NO-ECHO clause

The NO-ECHO 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

```
{NO-ECHO}
{NO ECHO}
{SECURE }
{OFF    }
```

Syntax rules

1. NO-ECHO, NO ECHO, SECURE and OFF are synonyms.

General rules

1. If a NO-ECHO clause is specified at the group level, it applies to each elementary input screen item in that group.
2. The NO-ECHO clause has an effect only during the execution of an ACCEPT Statement referencing the screen item.
3. The effect of the NO-ECHO clause is to prevent data corresponding to a screen item that has been specified

with the NO-ECHO 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.

OCCURS clause

The OCCURS clause describes repeated screen items and supplies information required for the application of subscripts.

General format

```
OCCURS Integer-6 TIMES
```

Syntax rules

1. The maximum number of dimensions for a table described in a screen description entry is two.
2. If a screen description entry includes the OCCURS clause, then if it or any item subordinate to it has a description that includes the TO, FROM, or USING clause, that screen description entry shall be part of a table with the same number of dimensions and number of occurrences in each dimension as the identifier representing the receiving or sending operand. The identifier representing the receiving or sending operand shall not be subordinate to an OCCURS clause with the DEPENDING phrase.
3. If a screen description entry that includes the OCCURS clause also contains the COLUMN clause, then the COLUMN clause shall include the PLUS or '-' phrase, unless the screen description entry also includes a LINE clause with a PLUS or '-' phrase.
4. If a screen description entry that includes the OCCURS clause also contains the LINE clause, then the LINE clause shall include the PLUS or '-' phrase, unless the screen description entry also includes a COLUMN clause with a PLUS or '-' phrase.

General rules

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 value of *Integer-6* represents the fixed number of occurrences of the subject of the entry.
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 '-' 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 '-' 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 '-' phrase, each occurrence behaves as though it had the same subordinate entries with the same LINE clause specified.

OUTPUT clause

The OUTPUT clause defines the alignment of the screen item for which it is specified. It has no effect on the actual storage of the item.

General format

```
OUTPUT {LEFT    }
       {RIGHT   }
       {CENTERED}
```

General rules

1. When the LEFT phrase is specified, leading spaces are removed and the screen item is left aligned.
2. When the CENTER phrase is specified, leading ad trailing spaces are removed and the screen item is centered.
3. When the RIGHT phrase is specified, trailing spaces are removed and the screen item is right aligned.

PROMPT clause

The PROMPT clause causes the trailing spaces or leading zeroes to be replaced by the prompt character '_'. It has no effect on the actual storage of the item.

General format

```
PROMPT [ CHARACTER IS Nonnumeric-Literal-1 ]
```

General rules

1. The CHARACTER phrase of the PROMPT clause is treated as a commentary.

REVERSE clause

The REVERSED clause causes the field to have reversed foreground and background colors.

General Format

```
{REVERSE-VIDEO}
{REVERSE      }
{REVERSED     }
```

SIGN clause

The SIGN clause specifies the position and the mode of representation of the operational sign.

General format

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

Syntax rules

1. The SIGN clause may be specified only for a screen description entry whose picture character-string contains the symbol 'S'.

General rules

1. The SIGN clause specifies the position and the mode of representation of the operational sign for the screen description entry to which it applies, or for each screen description entry subordinate to the group to which it applies.
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 screen description entry subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in that elementary entry takes precedence for that elementary entry.
4. 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. Valid signs for data items are '+' and '-'.
5. If the SEPARATE CHARACTER phrase is specified, 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 operational signs for positive and negative are the basic special characters '+' and '-', respectively.

SIZE clause

The SIZE clause specifies the number of character positions of the item on screen.

General format

```
SIZE IS {Data-Name-4}
        {Integer-7  }
```

Syntax rules

1. The SIZE clause may be specified only for elementary screen description entries.

TAB-GROUP clause

The TAB-GROUP clause specifies the handle of a Tab-Control where the screen entry will be added.

General format

```
TAB-GROUP IS {Tab-Control-Handle}
```

Syntax rules

1. Tab-Control-Handle is a USAGE HANDLE OF TAB-CONTROL data item.

TAB-GROUP-VALUE clause

The TAB-GROUP-VALUE clause specifies the index of the Tab-Control page where the screen entry will be added.

General format

```
TAB-GROUP-VALUE IS  {Data-Name-4}
                    {Integer-7  }
```

TO clause

The TO clause identifies the destination of the data in an ACCEPT screen Statement.

General format

```
TO Data-Name-5
```

Syntax rules

1. The category of *Data-Name-5* shall be a permissible category as a receiving operand in a MOVE Statement where the sending operand has the same PICTURE clause as the subject of the entry.
2. *Data-Name-5* shall be defined in the file, working-storage, local-storage, or linkage section. *Data-Name-5* shall not specify a variable-length group.
3. If the subject of this entry is subject to an OCCURS clause, *Data-Name-5* shall be specified without the subscripting normally required. Additional requirements are specified in the OCCURS clause.

General rules

1. The subject of the entry is an input screen item.
2. The TO clause has an effect only during execution of an ACCEPT screen Statement referencing the screen item.

UNDERLINED clause

The UNDERLINED clause causes the field to have the underscore video attribute applied to it.

General Format

```
{UNDERLINED}
{UNDERLINE }
```

General rules

1. If the UNDERLINED clause is specified and the character prompt is '_', the character prompt ' ' is assumed.

UPPER and LOWER clauses

The UPPER and LOWER clauses cause the content of the screen item for which they are specified to be converted to upper-case or to lower-case, respectively.

General format

```
{UPPER}
{LOWER}
```

Syntax rules

1. The UPPER or LOWER clause may be specified only for a screen description entry whose category is alphabetic, alphanumeric, or national.

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 {Data-Name-6}
      {Literal-2  }
```

Syntax rules

1. The category of the data item referenced by *Data-Name-6* shall be a permissible category as a receiving operand in a MOVE Statement where the sending operand has the same PICTURE clause as the subject of the entry.
2. The category of the data item referenced by *Data-Name-6* shall be a permissible category as a sending operand in a MOVE Statement where the receiving operand has the same PICTURE clause as the subject of the entry.
3. *Data-Name-6* shall be defined in the file, working-storage, local-storage, or linkage section. *Data-Name-6* shall not specify a variable-length group.
4. If the subject of this entry is subject to an OCCURS clause, *Data-Name-6* shall be specified without the subscripting normally required. Additional requirements are specified in the OCCURS clause.

General rules

1. If *Data-Name-6* is specified, specifying the USING clause is equivalent to specifying both the TO and FROM clauses, each specifying the same identifier.
2. If *Data-Name-6* is specified, the subject of the entry is both an input screen item and an output screen item.

3. If *Literal-2* is specified, specifying the `USING` clause is equivalent to specifying the VALUE clause.
4. If *Literal-2* is specified, the subject of the entry is an output screen item.

VALUE clause

The `VALUE` clause specifies the value of screen section displayable items.

General format

```
VALUE IS [{MULTIPLE}] {Data-Name-7}
         {TABLE   }  {Literal-3  }
```

Syntax rules

1. *Literal-3* shall be an alphanumeric literal. *Literal-3* shall not be a figurative constant.
2. MULTIPLE and TABLE are synonyms.

General rules

1. *Literal-3* specifies the value of the screen item that is displayed on the screen when directly or indirectly referenced by an ACCEPT screen Statement or a DISPLAY screen Statement.
2. The subject of the entry is an output screen item.
3. The MULTIPLE phrase is used only with certain control types that allow for multiple values.

ZERO-FILL clause

The ZERO-FILL phrase causes trailing spaces in the entered data to be replaced by the "0" character.

General Format

```
{ZERO-FILL   }
{NUMERIC-FILL}
```

Embedded Procedures

Embedded Procedures have been implemented to make programming even simpler. It is a way to attach a procedure, a paragraph or a section, to one or more screen section entries and have them activated before the control activation, after the control activation and when a function key is pressed.

Every Screen Section item, no matter if it is a group or a single control, can handle up to three different Embedded Procedures. If an Embedded Procedure is assigned to a group, it is inherited by all the groups or controls that it contains, unless they have their own embedded procedure set.

See Embedded Procedures in the User Interface manual for details.

Color management

There are two ways of managing colors.

- Using attributes
- Using RGB

See Color management in the User Interface manual for details.

Logical Record Concept

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

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:

- The grouping of logical records within the physical limitations of the file medium.
- The means by which the file can be identified.

File connector

A file connector is referenced by a file-name and it is a storage area that is not visible to the user that contains information used by the run unit to determine the status of input-output operations and of the connection to the physical file.

A file connector has several attributes that are specified by phrases and clauses in the file description entry, and the file control entry for the associated file-name and by the execution of I-O statements. These attributes are: organization (sequential, indexed, or relative); access mode (sequential, dynamic, or random); lock mode (automatic, manual, or none); locking mode (single record locking, multiple record locking, or none); in an open mode (input, output, i-o, extend); sharing mode (sharing with no other, sharing with read only, sharing with all other, or no sharing); and whether or not it is a report file connector. It also contains information about the file position indicator, the key of reference, the I-O status value, the current volume pointer, and file and record locks.

A file connector is either internal or external. For internal file connectors, one file connector is associated with each file description entry. For external file connectors, there is only one file connector that is associated with the run unit no matter how many file description entries describe the same file-name.

Conceptual Characteristics of a File

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

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

A physical record is a physical unit of information whose size and recording mode is convenient to a particular computer for the storage of data on an input or output device. The size of a physical record is hardware dependent and bears no direct relationship to the size of 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.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 Section, LOCAL-STORAGE Section and LINKAGE Section. Thus, WORKING-STORAGE Section, LOCAL-STORAGE Section and LINKAGE Section are 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, none of the clauses used to describe the data in the logical record have any effect on this transfer.

Record Concepts

The record description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name, if required, followed by a series of independent clauses, as required.

Concept of 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 two or more groups, etc. Thus, an elementary item may belong to more than one group.

Level-Numbers

A system of level-numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, level-numbers for records start at 01. Less inclusive data items are assigned higher (not necessarily successive) level-numbers not greater in value than 49. There are special level-numbers, 66, 77, 78, and 88, which 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 which are immediately subordinate to a given group item must be described using identical level-numbers greater than the level-number used to describe that group item.

Four 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.
4. Entries that specify constants.

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, which are not subdivisions of other items, and are not themselves subdivided, have been assigned the special level-number 77.

Entries that specify constant names have been assigned the special level-number 78.

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

Concept of Classes of Data

There are five categories of data items, described by the PICTURE clause. These are grouped into three classes: alphabetic, numeric, and alphanumeric. For alphabetic and numeric, the classes and categories are synonymous. The alphanumeric class includes the categories of alphanumeric edited, numeric edited, and alphanumeric (without editing). Every elementary item belongs to one of the classes and further to one of the categories. The class of a group item is treated as alphanumeric regardless of the class of elementary items subordinate to that group item. The following table depicts the relationship of the class and categories of data items.

Level of item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric edited Alphanumeric edited Alphanumeric
Group	Alphanumeric	Alphabetic Numeric Numeric edited Alphanumeric edited Alphanumeric

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 are represented to identify the sign of the item.

The `SIGN` clause permits the programmer to explicitly state the location of the operational sign.

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

Standard Alignment Rules

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

1. If the receiving data item is described as numeric:
    a. The data is aligned by decimal point and is moved to the receiving 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 paragraph 1a.
2. If the receiving data item is a 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.

3. If the receiving data item is alphanumeric (other than a numeric edited data item), alphanumeric edited, or alphabetic, the sending data is moved to the receiving character positions and aligned at the leftmost character position in the data item with space fill or truncation to the right, as required.

If the JUSTIFIED clause is specified for the receiving item, these standard rules are modified.

Item Alignment

Memory is organized in such a way that there are natural addressing boundaries (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.

Data items which are aligned on these natural boundaries are defined to be synchronized. Synchronization can 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 use of such items within a group may affect the results of statements in which the group is used as an operand.

Data Names

Uniqueness of Reference

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

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

Qualification

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

1. No other name has the identical spelling and hyphenation.
2. It is unique within the context of a `REDEFINES` clause.
3. The name exists within a hierarchy of names such that reference to the name can be made unique by mentioning one or more of the higher level names in the hierarchy.

These higher level names are called qualifiers and this process that specifies uniqueness is called qualification. Identical user-defined names may appear in a source program; however, uniqueness must then be established through qualification for each user-defined name explicitly referenced, except in the case of redefinition. 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 program would result in more than one occurrence of any of these special registers.

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

General formats

Format 1:

```
{Data-Name-1   } { {OF} Data-Name-2 } ... [ {OF} File-Name-1 ]
{Condition-Name}   {IN}                    {IN}
```

Format 2:

```
Paragraph-Name {OF} Section-Name
               {IN}
```

Format 3:

```
Text-Name {OF} Library-Name
          {IN}
```

Format 4:

```
LINAGE-COUNTER {OF} File-Name-2
               {IN}
```

Format 5:

```
PAGE-COUNTER {OF} Report-Name-2
             {IN}
LINE-COUNTER {OF} Report-Name-2
             {IN}
```

General rules

1. For each nonunique user-defined name that is explicitly referenced, uniqueness must be established through a sequence of qualifiers which precludes any ambiguity of reference.
2. A name can be qualified even though it does not need qualification; if there is more than one combination of qualifiers that ensures uniqueness, any such set can be used.
3. `IN` and `OF` are synonymous.
4. In Format 1, each qualifier must be the name associated with a level indicator, the name of a group item to which the item being qualified is subordinate, or the name of the conditional variable with which the condition-name being qualified is associated. Qualifiers are specified in the order of successively more inclusive levels in the hierarchy.
5. In *Format 1*, *Data-Name-1* or *Data-Name-2* may be a record-name.
6. If explicitly referenced, a paragraph name must not be duplicated within a section. When a paragraph name is qualified by a section name, the word `SECTION` must not appear. A paragraph name need not be qualified when referred to from within the same section.
7. *Text-Name* is the name of a copybook.
8. *Library-Name* is the name of a directory that contains *Text-Name*.

9. LINAGE-COUNTER must be qualified each time it is referenced if more than one file description entry containing a LINAGE clause has been specified in the source program.
10. LINE-COUNTER shall be qualified each time it is referenced in the procedure division if more than one report description entry is specified in the source element. In the report section, an unqualified reference to LINE-COUNTER is qualified implicitly by the name of the report in whose report description entry the reference is made. Whenever the LINE-COUNTER of a different report is referenced, LINE-COUNTER shall be qualified explicitly by the report-name associated with the different report.
11. PAGE-COUNTER shall be qualified each time it is referenced in the procedure division if more than one report description entry is specified in the source element. In the report section, an unqualified reference to the PAGE-COUNTER is qualified implicitly by the name of the report in whose report description entry the reference is made. Whenever the PAGE-COUNTER of a different report is referenced, PAGE-COUNTER shall be qualified explicitly by the report-name associated with the different report.

Report counters

The PAGE-COUNTER and LINE-COUNTER identifiers are generated automatically and exist independently for each report.

General format

```
{ PAGE-COUNTER } { OF } report-name-1
{ LINE-COUNTER } { IN }
```

Syntax rules

1. In the report section, PAGE-COUNTER and LINE-COUNTER may be referenced only in a SOURCE clause. In the procedure division, PAGE-COUNTER and LINE-COUNTER may be referenced in any context where an integer data item may appear.

   NOTE: Because each report maintains an independent PAGE-COUNTER and LINE-COUNTER, it is the programmer's responsibility to assign the correct values to any page numbers and to ensure that report groups are printed correctly within the limits of the page.
2. LINE-COUNTER shall not be referenced as a receiving operand.

General rules

1. PAGE-COUNTER and LINE-COUNTER reference temporary unsigned integer data items of class and category numeric, which are maintained for each report.
2. The initial value of PAGE-COUNTER is set to 1 by the execution of an INITIATE statement for the corresponding report and its value is updated by 1 during each page advance. It is reset to 1 when a report group that contains a NEXT GROUP clause with a RESET phrase is printed.
3. The initial value of LINE-COUNTER is set to zero by execution of an INITIATE statement for the corresponding report. It is reset to zero whenever a page advance takes place.
4. At the time each report line is printed, the value of LINE-COUNTER specifies the line number of the page on which the line is printed. The value of LINE-COUNTER after the printing of a report group is the same as the line number of the last line printed, unless a NEXT GROUP clause is defined for the report group, in which case the final value of LINE-COUNTER is defined by the general rules for the NEXT GROUP clause.
5. The values of PAGE-COUNTER and LINE-COUNTER are not affected by the processing of a dummy report group, nor by the processing of a report group whose printing is suppressed by means of the SUPPRESS statement.

Subscripting

Subscripts are used when reference is made to an individual element within a table of like elements that have not been assigned individual data-names.

General format

```
{Data-Name-1   } ( { {Integer-1   } [ {+} {Integer-2   } ] ... } ... )
{Condition-Name}     {Data-Name-2 }   {-} {Data-Name-3 }
                   {  ALL                                 }
                   {  Index-Name-1                        }
```

Syntax rules

1. The data description entry containing *Data-Name-1* or the data name associated with *Condition-Name* must contain an OCCURS clause or must be subordinate to a data description entry which contains an OCCURS clause.
2. Except as defined in syntax rule 3, when a reference is made to a table element, the number of subscripts must equal the number of OCCURS clauses in the description of the table element being referenced. When more than one subscript is required, the subscripts are written in the order of successively less inclusive dimensions of the table.
3. Each table element reference must 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
4. *Data-name-2* and *Data-name-3* may be qualified and must be numeric elementary items representing an integer.
5. *Integer-1* and *Integer-2* may be signed and, if signed, they must be positive.
6. *Index-Name-1* is a data description entry in the hierarchy of the table being referenced which contains an INDEXED BY phrase specifying the index name.

General rules

1. The value of the subscript must be a positive integer. The lowest possible occurrence number represented by a subscript is 1. The first element of any given dimension of a table is referenced by an occurrence number of 1. Each successive element within that dimension of the table is referenced by occurrence numbers of 2, 3, ...The highest permissible occurrence number for any given dimension of the table is the maximum number of occurrences of the item as specified in the associated OCCURS clause.
2. The index name ALL is used as a function argument for a function that allows a variable number of arguments. It can be used only when the subscripted identifier is used as a function argument and can not be used when Condition-Name is specified. The index name ALL causes all occurrences of Data-Name-1 to be passed to the function.

Reference Modification

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

General Format

```
Data-Name ( Leftmost-Character-Position : [Length] )
```

Syntax Rules

1. *Data-name* is a Data Item, as defined in the Definitions section in the Preface of this document.
2. *Leftmost-Character-Position* and Length must be arithmetic expressions.
3. Unless otherwise specified, reference modification is allowed anywhere an identifier referencing a data item of the class alphanumeric is permitted.
4. *Data-Name* may be qualified or subscripted.

General Rules

1. Each character of a data item referenced by *Data-Name* is assigned an ordinal number incrementing by one from the leftmost position to the rightmost position. The leftmost position is assigned the ordinal number one. If the data description entry for *Data-Name* contains a `SIGN IS SEPARATE` clause, the sign position is assigned an ordinal number within that data item.
2. If the data item referenced by *Data-Name* is described as numeric, numeric edited, alphabetic, or alphanumeric edited, it is operated upon for purposes of reference modification as if it were redefined as an alphanumeric data item of the same size as the data item referenced by *Data-Name*.
3. Reference modification for an operand is evaluated as follows:
    a. If subscripting is specified for the operand, the reference modification is evaluated immediately after evaluation of the subscripts.
    b. If the subscripting is not specified for the operand, the reference modification is evaluated at the time subscripting would be evaluated if subscripts had been specified.
4. Reference modification creates a unique data item which is a subset of the data item referenced by *Data-Name*. This unique data item is defined as follows:
    a. The evaluation of *Leftmost-Character-Position* specifies the ordinal position of the leftmost character of the unique data item in relation to the leftmost character of the data item referenced by *Data-Name*. Evaluation of *Leftmost-Character-Position* must result in a positive nonzero integer less than or equal to the number of characters in the data item referenced by *Data-Name*.
    b. The evaluation of length specifies the size of the data item to be used in the operation. The evaluation of length must result in a positive nonzero integer. The sum of *Leftmost-Character-Position* and length minus the value one must be less than or equal to the number of characters in the data item referenced by *Data-Name*. If length is not specified or is zero, the unique data item extends from and includes the character identified by *Leftmost-Character-Position* up to and including the rightmost character of the data item referenced by *Data-Name*.
5. The unique data item is considered an elementary data item without the `JUSTIFIED` clause. It has the same class and category as that defined for the data item referenced by *Data-Name* except that the categories numeric, numeric edited, and alphanumeric edited are considered class and category alphanumeric.

Condition-name

If explicitly referenced, a condition-name must be unique or be made unique through qualification and/or subscripting.

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

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

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

ADDRESS OF

The ADDRESS OF function returns the pointer to a data item.

General Format

```
ADDRESS OF Data-Name
```

Syntax Rules

1. *Data-name* is a Data Item, as defined in the Definitions section in the Preface of this document.

CURRENT-DATE

The CURRENT-DATE function returns the current date as a eigth characters string. It can always be used where a string literal is allowed. It requires -cv compiler option.

General Format

```
CURRENT-DATE
```

General Rules

1. The date is returned in the format "MM/DD/YY", where MM is the month number (1-12), DD is the day number (1-31) and YY is the year number stripped of the century (0-99).

EXCEPTION-OBJECT

EXCEPTION-OBJECT is a reference to a java.lang.Throwable object that is automatically set by the runtime each time an exception occurs.

General Format

```
EXCEPTION-OBJECT:>methodName
```

Syntax Rules

1. Consult the java.lang.Throwable for a list of available methods and types.

FUNCTION

The result of a function depends of the function name.

Format 1

```
FUNCTION Function-Name
```

Format 2

```
$Function-Name
```

Syntax Rules

1. In Format 2 there must be no space between the dollar sign and the function name.
2. Consult Intrinsic Functions for a list of available functions and their type.

LENGTH OF

The `LENGTH OF` function returns the size in bytes of a data item. It can always be used where a numeric literal is allowed.

General Format

```
LENGTH OF Data-Name
```

Syntax Rules

1. *Data-name* is a Data Item, as defined in the Definitions section in the Preface of this document.
2. *Data-Name* may be qualified.
3. *Data-Name* may not be reference modified.

General Rules

1. If *Data-Name* is a national item, then the number of digits is returned instead of the number of bytes.
2. If *Data-Name* specifies the OCCURS clause, then the size of the first occurrence is returned.

LINAGE-COUNTER

A separate LINAGE-COUNTER special register is generated for each FD entry containing a LINAGE clause.

General Format

```
LINAGE-COUNTER
```

General Rules

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

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

3. LINAGE-COUNTER is automatically modified by any WRITE statement for this file.

PROGRAM-ID

The PROGRAM-ID special register returns the name of the current program.

General Format

```
PROGRAM-ID
```

General Rules

1. This syntax is supported only in RM compatibility mode, so the -cr compiler option must be used.

2. PROGRAM-ID contains the name of the class file in which the program is implemented, stripped of the class extension.

TIME-OF-DAY

The TIME-OF-DAY function returns the current time as a six digit number. It can always be used where a numeric literal is allowed. It requires -cv compiler option.

General Format

```
TIME-OF-DAY
```

General Rules

1. The time is returned in the format HHNNSS, where HH is the number of hours (0-23), NN is the number of minutes (0-59) and SS is the number of seconds (0-59).

RECORD-POSITION

The `RECORD-POSITION` function returns the character position of a data item in a record description, starting at 1. It can always be used where a numeric literal is allowed.

General Format

```
RECORD-POSITION OF Data-Name
```

Syntax Rules

1. *Data-name* is a Data Item, as defined in the Definitions section in the Preface of this document.

2. *Data-Name* may be qualified.

3. *Data-Name* may not be reference modified.

4. If *Data-Name* is subordinate to an OCCURS clause, then it must be referenced with subscripting or indexing.

5. If *Data-Name* refers to a table item, the value is computed from the first occurrence of that item.

WHEN-COMPILED

The WHEN-COMPILED option returns the compile time timestamp as string. It requires either -cr or -cv compiler options.

General Format

```
WHEN-COMPILED
```

General Rules

1. Under -cr option the timestamp is returned in the format "HH.NN.SS MMM DD, YYYY" where HH is the number of hours (0-23), NN is the number of minutes (0-59), SS is the number of seconds (0-59), MMM is the month name in the current locale, DD is the day number (1-31) and YYYY is the year number. Example: 16.13.53Jun 18, 2019.

1. Under -cv option the timestamp is returned in the format MM/DD/YYHH.NN.SS, where MM is the month number (1-12), DD is the day number (1-31), YY is the year number stripped of the century (0-99), HH is the number of hours (0-23), NN is the number of minutes (0-59) and SS is the number of seconds (0-59). Example: 06/18/1916.13.53

Chapter 5

Procedure Division

The procedure division in a method, or a program contains procedures to be executed.

The procedure division in an instance definition and a factory definition contains the methods that may be invoked on the instance object or factory object.

The procedure division in an interface contains method prototypes.

The procedure division in a function prototype, method prototype, or program prototype specifies the parameters, any returning item, and any exceptions that may be raised.

Procedure division structure

General format

```
PROCEDURE DIVISION

  [ { USING [{BY VALUE    }] {Data-Name-1}... } ]
           {BY REFERENCE}

    { CHAINING {Data-Name-1}...               }

  [ RETURNING Data-Name-2 ]

  [ RAISING {Class-Name-1    } ... ] .
            {Interface-Name-1}

  [ DECLARATIVES.

    { Section-Name-1 SECTION .

      Use-Statement.

      [Sentence-1] ... [ [ Paragraph-Name-1 . ] [Sentence-2] ... ] ... } ...

    END-DECLARATIVES . ]

  [ [ Section-Name-2 SECTION . [Sentence-3] ... [ [ Paragraph-Name-2 . ] [Sentence-
   3] ... ] ... ] ] ...
    [ [ Paragraph-Name-3 . ] [Sentence-
   4] ...                                                  ]
```

Syntax rules

1. *Data-Name-1*, *Data-Name-2*, *Class-Name-1*, *Interface-Name-1*, *Section-Name-1*, *Section-Name-2*, *Paragraph-Name-1*, *Paragraph-Name-2*, *Paragraph-Name-3* and *Paragraph-Name-4* are User-defined words, as defined in the Definitions section in the Preface of this document.
2. *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*.
3. The `RETURNING` phrase may be specified only in a method definition.
4. The `RAISING` phrase may be specified only in a method definition.
5. *Class-Name-1* shall be the name of a class specified in the REPOSITORY Paragraph in the Configuration Section of the `ENVIRONMENT DIVISION`.
6. *Interface-Name-1* shall be the name of a class specified in the REPOSITORY Paragraph in the Configuration Section of the `ENVIRONMENT DIVISION`.
7. `BY VALUE` and `BY REFERENCE` are treated as a commentary.
8. If `USING` is used, *Data-Name-1* must be declared in the program Linkage Section. If `CHAINING` is used, *Data-Name-1* must be declared in the program Working-Storage Section or File Section.
9. If *Data-Name-1* is a table and is referenced without subscripting:

- If USING is used, *Data-Name-1* is treated as a buffer whose size is the size of each table element multiplied by the number of occurrences. For example, the following code:

```
linkage section.
01 par occurs 2.
   03 para pic x.
   03 parb pic x.
procedure division using par.
main.
   display para(1).
   display parb(1).
   display para(2).
   display parb(2).
```

is equivalent to:

```
linkage section.
01 par pic x(4).
procedure division using par.
main.
   display par(1:1).
   display par(2:1).
   display par(3:2).
   display par(4:2).
```

- If CHAINING is used, the first element of *Data-Name-1* is considered. For example, the following code:

```
working-storage section.
01 par occurs 2.
   03 para pic x.
   03 parb pic x.
procedure division chaining par.
```

- is equivalent to:

```
working-storage section.
01 par occurs 2.
   03 para pic x.
   03 parb pic x.
procedure division chaining par(1).
```

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 formal parameters used by the method or program for any arguments passed to it. The arguments passed from the activating element are:

- the arguments specified in the USING phrase of a CALL Statement
- the arguments specified in the USING phrase of a CHAIN Statement
- the arguments specified in the USING phrase of an INVOKE Statement
- the arguments specified in an inline invocation of a method

The correspondence between the arguments and the formal parameters is established on a positional basis.

The conformance requirements for formal parameters and returning items are specified in Conformance for parameters and returning items.

3. *Data-Name-2* is the name used in the function, method, or program for the result that is returned to the activating element according to the Results of runtime element execution.
4. If the argument is passed by reference, the activated runtime element operates as if the formal parameter occupies the same storage area as the argument.
5. If the argument is passed by content, the activated runtime element operates as if the record in the linkage section were allocated by the activating runtime element 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 runtime element. That argument is moved to this allocated record without conversion. This record is then treated by the activated runtime element as if it were the argument and as if it were passed by reference.

   If the activated runtime element is a method, then this allocated record is

- a data item of the same category, usage, and length as the argument, if the formal parameter is described with the `ANY LENGTH` clause,
- otherwise, a data item with the same description and the same number of bytes as the formal parameter, where the maximum length is used if the formal parameter is described as a variable-occurrence data item.

   The argument is used as the sending operand and the allocated record 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 index, object, or pointer, a SET Statement
- otherwise, a MOVE Statement.

   The allocated record is then treated as if it were the argument and it were passed by reference.
6. 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.
7. If *class-name-1* is specified, an object of *class-name-1* may be raised by a RAISE Statement within this element.
8. If *interface-name-1* is specified, an object that implements *interface-1* may be raised by a RAISE Statement within this element.
9. The CHAINING phrase is used only for a program that is the main program executed in a run unit.
   a. If the program is initiated from the host system, each parameter is initialized to the corresponding command line argument.
   b. If the program is initiated by a CHAIN statement, then each parameter receives the value of the corresponding USING item specified by that CHAIN statement.
   c. Values are assigned to each parameter as if the value were the alphanumeric source for an elementary MOVE to parameter.
   d. If there are fewer arguments than parameters, then the excess parameters are initialized according to the rules that would apply if they were not listed in the CHAINING phrase.
   e. If there are more arguments than parameters, the excess arguments are ignored.

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 keyword `DECLARATIVES` and followed by the keywords `END DECLARATIVES`.

The sections specified between the keywords `DECLARATIVES` and `END DECLARATIVES` constitute the declarative portion of a source element. All other sections in a source element constitute the nondeclarative portion.

Arithmetic expressions

Arithmetic expressions are used as operands of certain conditional and arithmetic statements.

An arithmetic expression can consist of any of the following:

1. An identifier described as a numeric elementary item (including numeric functions)
2. A numeric literal
3. The figurative constant ZERO
4. An OBJECT-REFERENCE of Java numeric primitive types (i.e. "int") or objects (i.e. "java.lang.Integer")
5. Identifiers and literals, as defined in items 1, 2, and 3, separated by arithmetic operators
6. Two arithmetic expressions, as defined in items 1, 2, 3, or 4, separated by an arithmetic operator
7. An arithmetic expression, as defined in items 1, 2, 3, 4, or 5, enclosed in parenthesis

Any arithmetic expression can be preceded by a unary operator.

Identifiers and literals that appear in arithmetic expressions must represent either numeric elementary items or numeric literals on which arithmetic can be performed.

Arithmetic operators

Arithmetic operators are represented by specific characters that must be preceded and followed by a space.

The following binary arithmetic operators can be used in arithmetic expressions:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

The following unary arithmetic operators can be used in arithmetic expressions:

Operator	Meaning

+	Multiplication by +1
-	Multiplication by -1

Parenthesis can be used in arithmetic expressions to specify the order in which elements are to be evaluated.

Expressions within parenthesis are evaluated first. When expressions are contained within nested parenthesis, evaluation proceeds from the least inclusive to the most inclusive set.

When parenthesis are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchic order is implied:

1. Unary operator
2. Exponentiation
3. Multiplication and division
4. Addition and subtraction

Parenthesis either eliminate ambiguities in logic where consecutive operations appear at the same hierarchic level, or modify the normal hierarchic sequence of execution when this is necessary. When the order of consecutive operations at the same hierarchic level is not completely specified by parenthesis, the order is from left to right.

An arithmetic expression can begin only with a left parenthesis, a unary operator, or an operand (that is, an identifier or a literal). It can end only with a right parenthesis or an operand. An arithmetic expression must contain at least one reference to an identifier or a literal.

There must be a one-to-one correspondence between left and right parenthesis in an arithmetic expression, with each left parenthesis placed to the left of its corresponding right parenthesis.

If the first operator in an arithmetic expression is a unary operator, it must be immediately preceded by a left parenthesis if that arithmetic expression immediately follows an identifier or another arithmetic expression.

Pointer arithmetic

A pointer is an address, which is a numeric value. Therefore, you can perform arithmetic operations on a pointer just as you can on a numeric value.

For example, the following statements produce a pointer that points one character position past the beginning of WRK-ITEM1:

```
set ptr-1 to address of wrk-item1.
compute ptr-1 = ptr-1 + 1.
```

For a full support of pointer arithmetic operations, compile the program with the -cp option.

Conditional expressions

A conditional expression causes the object program to select alternative paths of control, depending on the truth value of a test. Conditional expressions are specified in EVALUATE, IF, PERFORM, and SEARCH statements.

Class condition

The class condition determines whether the content of a data item is alphabetic or numeric.

Syntax.

```
identifier-1  [IS]  [NOT]  {NUMERIC}
                           {ALPHABETIC}
                           {POSITIVE}
                           {NEGATIVE}
                           {ALPHABETIC_LOWER}
                           {ALPHABETIC_UPPER}
                           {HANDLE OF} identifier-2
```

Relation conditions

A relation condition specifies the comparison of two operands. The relational operator that joins the two operands specifies the type of comparison. The relation condition is true if the specified relation exists between the two operands; the relation condition is false if the specified relation does not exist.

Syntax

```
operand-1  [ IS ]  [ NOT ]  { GREATER THAN               }               operand-2
                            { LESS THAN                  }
                            { EQUAL TO                   }
                            { >                          }
                            { <                          }
                            { <>                         }
                            { =                          }
                            { >=                         }
                            { <=                         }
                            { LIKE [ TRIMMED { LEFT  } ] }
                                             { RIGHT }
                                  [ CASE-INSENSITIVE  ]
                                  [ CASE-SENSITIVE    ]
                                  [ APPROX { numeric-literal-1   } ]
                                           { numeric-data-item-1 }
```

This additional syntax is supported when compiling with -cm:.

```
operand-1 { EQUALS        }  operand-2
          { IS UNEQUAL TO }
          { EXCEEDS       }
```

- *operand-1* and *operand-2* may be either an arithmetic expression, a data name, or a literal.
- When the LIKE operator is used, *operand-2* identifies a regular expression.
- After the APPROX clause an integer value must be specified, specifying the maximum number of allowed errors, computed according to the Levenshtein distance. The second operator in this case cannot be a regular expression, however it can contain the wildcard "?", any character, and "*", a sequence of 0 or more characters. "\" is the escape character, so that: "*" becomes *, "\?" becomes "?" and "\\" becomes "\".

Condition-name condition

A condition-name condition tests a conditional variable to determine whether its value is equal to any values that are associated with the condition-name.

A condition-name is used in conditions as an abbreviation for the relation condition. The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

If condition-name-1 has been associated with a range of values (or with several ranges of values), the conditional variable is tested to determine whether its value falls within the ranges, including the end values. The result of the test is true if one of the values that corresponds to the condition-name equals the value of its associated conditional variable.

The following example illustrates the use of conditional variables and condition-names:

```
01  FILE-STATUS      PIC  XX.
    88  SUCCESS  VALUE "00" THRU "09".
    88  FAILED   VALUE "10" THRU "99".
```

FILE-STATUS is the conditional variable; SUCCESS and FAILED are condition-names.

For individual records in the file, only one of the values specified in the condition-name entries can be present.

The following IF statements can be added to the above example to determine the file status:

IF SUCCESS... (Tests for values "00" through "09")

IF FAILED... (Tests for values "10" through "99")

Depending on the evaluation of the condition-name condition, alternative paths of execution are taken by the object program.

Complex condition

A complex condition is formed by combining simple conditions, combined conditions, or complex conditions with logical operators, or negating those conditions with logical negation.

Each logical operator must be preceded and followed by a space. The following table shows the logical operators and their meanings.

Operator	Meaning
AND	Logical conjunction. The truth value is true when both conditions are true.
OR	Logical inclusive. The truth value is true when either or both conditions are true.
NOT	Logical negation. Reversal of truth value (the truth value is true if the condition is false.

Unless modified by parentheses, the following is the order of precedence (from highest to lowest):

1. Arithmetic operations
2. Simple conditions
3. NOT
4. AND
5. OR

The truth value of a complex condition (whether parenthesized or not) is the truth value that results from the interaction of all the stated logical operators on either of the following:

- The individual truth values of simple conditions
- The intermediate truth values of conditions logically combined or logically negated

A complex condition can be either of the following:

- A negated simple condition
- A combined condition (which can be negated)

Combined condition

Two or more conditions can be logically connected to form a combined condition.

Syntax

```
condition-1  {AND}  condition-2
             {OR }
```

The condition to be combined can be any of the following:

- A simple-condition
- A negated simple-condition
- A combined condition
- A negated combined condition (that is, the NOT logical operator followed by a combined condition enclosed in parentheses)

When relation-conditions are written consecutively, any relation-condition after the first can be abbreviated in one of two ways:

- Omission of the subject
- Omission of the subject and relational operator

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. A procedure-name is a word used to refer to a paragraph or section in the source element 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 keywords `END DECLARATIVES`, and ends with a separator period.

Paragraphs

A paragraph consists of a paragraph-name followed by a separator period and by zero, one, or more successive sentences or, if the paragraph-name is omitted, one or more successive sentences following the procedure division header or a section header. 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 keywords `END DECLARATIVES`, and ends with a separator period.

Procedural statements and sentences

A procedural statement is a unit of the COBOL language that specifies an action to be taken. Statements are described in Procedure Division Statements.

Within the procedure division, there are the following types of statements:

- declarative statements, which specify actions that may be taken during the processing of other statements
- imperative statements, which specify unconditional actions
- conditional statements, which specify, or contain one or more phrases that specify, actions that depend on the truth value of a condition.

A declarative statement begins with the statement name USE and directs that actions be taken in response to specified conditions encountered during the processing of other statements.

An imperative statement specifies an unconditional action to be taken by the runtime element or is a conditional statement that is delimited by its explicit scope terminator.

A conditional statement specifies that the truth value of a condition is evaluated and used to determine subsequent flow of control. Any statement with a conditional phrase that is not terminated by its explicit scope terminator is a conditional statement.

A sentence is a sequence of one or more procedural statements, the last of which is terminated by a separator period.

Wherever 'imperative-statement' appears in the general format of a statement, 'imperative-statement' refers to one or more imperative statements ended either by a separator period or by any phrase associated with that general format.

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.

Scope of statements

A procedure division statement's scope is the range of words and separators that constitute the statement. The scope of a statement begins with the first word of the statement name and continues until the statement is either explicitly or implicitly terminated, and includes any procedural statements occurring syntactically between the start and termination of that statement.

Explicit scope termination

The scope of a statement may be explicitly terminated by using its associated scope terminator. A statement written with its explicit scope terminator is a subset of imperative statements and is termed a delimited scope statement.

An explicit scope terminator terminates the scope of:

1. the most-recently preceding unterminated statement having the statement-name for which that scope terminator is defined, and
2. any unterminated statements that appear between that statement-name and the explicit scope terminator.

Implicit scope termination

The scope of a statement that is not explicitly terminated is implicitly terminated as follows:

1. for a single imperative statement not contained within another statement, by
    a. any element that follows the exhaustion of the statement's syntax, or
    b. the next-encountered statement-name, or
    c. a separator period.
2. for a single imperative statement contained within another statement, by
    a. any element that terminates an imperative statement not contained within another statement,
    b. the termination of the scope of any containing statement, or
    c. the next phrase of any containing statement.
3. for a conditional statement not contained within another statement, by a separator period.
4. for a conditional statement contained within another statement, by
    a. the termination of any containing statement, or
    b. the next phrase of any containing statement.

Any phrase encountered is the next phrase of the most-recently preceding unterminated statement with which that phrase may be syntactically associated. If all permitted occurrences of a phrase have already been specified for a given statement, a subsequent occurrence of that phrase is not syntactically associated with that statement. An unterminated statement is any statement that has begun but has not yet been either explicitly or implicitly terminated.

A contained statement is also referred to as a nested statement.

Execution

Run unit organization

At runtime, the highest-level unit of a COBOL application is the run unit. A run unit is an independent entity that may be executed without communicating with, or being coordinated with, any other run unit except that it may process files or set and test switches that were written or will be read by other run units. A run unit contains one or more runtime modules.

A runtime module results from compiling a compilation unit. Each runtime module contains one or more runtime elements.

A runtime element results from the compilation of a method or program. When a runtime element is activated, parameters upon which it is to operate may be passed to it by the runtime element that calls it.

A run unit and each of its contained runtime modules may also contain resources and data storage areas needed for the execution and intercommunication of the runtime elements contained in the run unit.

A run unit may additionally contain runtime modules and data storage areas derived from the compilation of compilation units written in languages other than COBOL.

State of a method, object, or program

State of a method or program

The state of a method or program at any point in time in a run unit may be active or inactive. When a function, method, or program is activated, its state may also be initial or last-used.

Active state

A method or program may be activated recursively. Therefore, several instances of a method or program may be active at once. When a rule indicates that the active state of a runtime element is tested, it is in the active state if any instance of the element is active.

An instance of a method is placed in an active state when it is successfully activated and remains active until the execution of a GOBACK, STOP or EXIT PROGRAM 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 runtime element. 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 runtime element.

Whenever an instance of a method or program is activated, the control mechanisms for all PERFORM Statements contained in that instance of the function, method, or program are set to their initial states.

Initial and last-used states of data

When a 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 is placed in the initial state every time the method or program in which it is described is activated.

  Static data is placed in the initial state:

  i. The first time the method or program in which it is described is activated in a run unit.
  ii. 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.
  iii. 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.

When data in a method or program is placed in the initial state, the following occurs:

i. The internal data described in the working-storage section is initialized as described in the VALUE clause.

ii. The method or program's internal file connectors are initialized by setting them to not be in any open mode.

iii. The attributes of screen items are set as specified in the screen description entry.

iv. For each dynamic-capacity table, except where the table is defined by an elementary entry with a `VALUE` clause, the capacity of the table is set to the minimum capacity specified in the corresponding `OCCURS` clause. If the `INITIALIZED` keyword is present in the `OCCURS` clause, all the occurrences, if any, of the table are then initialized.

v. The length of each any-length elementary item that is specified without a `VALUE` clause is set to zero, before data is placed in the initial state.

- 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.

Initial state of object data

The initial state of an object is the state of the object immediately after it is created. Internal data, internal file connectors, and the attributes of screen items are initialized in the same manner as when data in a method or program is placed in the initial state, in accordance with "Initial state" above.

Explicit and implicit transfers of control

The flow of the control mechanism transfers control from statement to statement in the sequence in which they were written 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 such as PERFORM, USE, and SORT, 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 Statement is executed, an implicit transfer of control occurs to any associated input or output procedures.
3. When any I/O 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.

If control is transferred either implicitly or explicitly to a paragraph containing no statements, execution proceeds as if the paragraph contained only a single sentence consisting of a CONTINUE Statement.

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 execution of an EXIT PROGRAM, GOBACK, or STOP Statement transfers control outside the COBOL source element.

In the declarative section, there is no next executable statement after either the last statement when the paragraph in which it appears is not being executed under the control of some other COBOL statement, or the last statement 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 also not the last statement of the procedure that is the exit of the active PERFORM Statement. In these cases the flow of control is undefined.

There is also no next executable statement after 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, after the end marker, and if there are no procedure division statements in a program, function, or method. In these cases, an implicit GOBACK Statement without any optional phrases is executed.

Item identification

Item identification is the process of identifying a specific data item referenced by an identifier by evaluating all of that identifier's references. If a step in the evaluation of an identifier requires evaluation of another identifier or an arithmetic expression, that evaluation is done in full before proceeding to the next step. The item identification steps that are applicable to that identifier are evaluated in the following order:

1. function evaluation
2. inline method invocation
3. subscript evaluation
4. object property evaluation
5. length evaluation for an occurs-depending group item
6. reference modification

Unless otherwise specified, item identification is done for an identifier as the first step in the evaluation of that identifier and the identifiers within a statement are evaluated in left to right order as the first operation of the execution of that statement.

Multiple items must be separated by space or by comma. If DECIMAL-PONT IS COMMA appears in the Special-Names, it's necessary to put a space after each comma when separating numeric constant values, or the following misbehavior will occur:

```
1,2
```

is treated as 1,2 decimal value instead of two different numeric values 1 and 2.

Results of runtime element execution

The result of the execution of a program, function, or method that specifies a RETURNING phrase in its procedure division header, is the content of the data item referenced by that RETURNING phrase.

The result becomes available to the activating element after the activated element returns as follows:

- If the runtime element is activated by a CALL or INVOKE Statement, the result is placed in the data item referenced by that RETURNING phrase of that activating statement.
- If the runtime element is activated by an inline method invocation, the result is placed in the temporary data item referenced by that identifier.

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.

Alignment of data within data items

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 specified by the IEEE (Institute of Electronics and Electrical Engineers) Standard 754.
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 a floating-point numeric-edited data item and the value to be edited is not zero, the data moved to the edited data item is aligned so that the leftmost digit is not zero.
5. 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 the JUSTIFIED clause.

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 be defined or undefined 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.

Normal run unit termination

When normal run unit termination occurs, the runtime system performs the following:

1. An implicit CLOSE Statement without any phrases is executed for each file that is in the open mode. These implicit CLOSE Statements shall be executed for all open files in the run unit, even when an error occurs during the execution of one or more of the CLOSE Statements. Any declaratives associated with these files are not executed.
2. Any storage obtained with an M$ALLOC routine and not yet released by a M$FREE routine is released.
3. All instance objects are destroyed.

   NOTE - Any open files in an object are closed before the object is deleted.

4. Any resources occupied by dynamic capacity tables or any-length elementary items are freed.

Abnormal run unit termination

When abnormal run unit termination occurs, the runtime system attempts to perform the operations of normal termination as specified in 1"Normal run unit termination" above. The circumstances of abnormal termination may be such that execution of some or all of these operations is not possible. The runtime system performs all operations that are possible.

The operating system shall indicate an abnormal termination of the run unit if such a capability exists within the operating system.

Condition handling

Exception conditions

An exception condition is either a condition associated with a specific exception status indicator or an exception object. An exception object is any object that is raised by the execution of either a RAISE Statement in which the object is specified in the `RAISING` phrase. An exception status indicator is a conceptual entity that exists for each function, method, or program for each exception condition and has two states, set or cleared. The initial state of all exception status indicators is cleared. An exception status indicator is set when the associated exception exists. Associated with each exception status indicator are one or more exception-names. These exception names, together with the interface-names and class-names of exception objects are used to enable checking for the exception condition, to specify the action to be taken when the exception is raised, and to determine which exception condition caused an exception declarative to be executed.

Unless otherwise specified, if more than one exception is detected during the execution of a statement, the one that is set to exist is undefined. The execution of the statement may be successful or unsuccessful, depending on the rules for the statement, and the fatality of the exception condition. If no exception is detected during the execution of a statement or if checking for an exception that occurs is not enabled, no exception condition is raised. All exception status indicators are cleared at the beginning of the execution of any statement.

Normal completion of a declarative procedure

A declarative procedure is said to complete normally if, during execution of the declarative procedure, none of the following occur:

1. An EXIT PROGRAM, GOBACK, RESUME, or STOP Statement that is specified in this method or program is executed within the scope of the declarative.
2. Any directly or indirectly activated runtime element terminates the run unit.

Fatal exception conditions

If a fatal exception condition exists, processing of the statement is interrupted and one of the following occurs in the order specified:

1. If a conditional phrase without the NOT phrase is specified in the interrupted statement and the rules for that statement indicate that the fatal exception condition is to be processed by the conditional phrase, the procedures for non-fatal exceptions as specified in "Non-fatal exception conditions" below, apply.
2. If there is an applicable USE Statement in the source unit, the associated declarative is executed.
3. If the exception condition is neither Exit Program nor Goback, and there is an applicable PROPAGATE Directive, the exception condition is propagated.
4. Execution of the run unit is terminated abnormally as specified in "Abnormal run unit termination" above.

Non-fatal exception conditions

If a non-fatal exception condition other than an exception object is set to exist, processing of the statement is interrupted and one of the following occurs in the order specified:

1. If a conditional phrase without the NOT phrase is specified in the interrupted statement, the imperative statement associated with that conditional phrase is executed as specified in the rules for the specific statement.
2. Execution of the statement continues as specified in the rules for that statement.

Exception objects

When an exception object is raised, the following occurs:

1. The predefined object reference EXCEPTION-OBJECT is set to the content of the object reference specified in the RAISE Statement that caused the exception object to be raised.
2. The last exception status is set to indicate that an exception object has been raised.

If an exception object is raised by a RAISE Statement, the method execution terminates and the exception condition is cached by the TRY Statement of the caller.

Incompatible data

Incompatible data exists when the content of a sending operand is not valid only in the following cases:

1. When a numeric-edited data item is the sending operand of a de-editing MOVE Statement and the content of that data item is not a possible result for any editing operation in that data item, the result of the MOVE operation is undefined.
2. When the internal format of an any-length elementary item is not correctly formed or does not agree with the corresponding ANY LENGTH clause.

3. When the internal format of a dynamic-capacity table, as defined by the implementor, is not correctly formed or does not agree with the corresponding OCCURS clause.

If the content of a sending operand is not referenced by a given execution of a statement, any incompatible data in that operand is not detected. If part of a sending operand's content is referenced by a given execution of a statement, it is undefined whether any incompatible data in the unreferenced content is detected.

NOTE - The content of a data item is not referenced when the data item is a receiving operand, unless that data item is also a sending data item.

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.

At end condition

The at end condition is associated with a sort file or the I-O status for a file connector. For sort files, the at end condition is set to exist when the sort operation has returned all of the records that were sent to it and there are no more records to be sorted. It no longer exists when the execution of the SORT operation referencing the sort file terminates. For other files, the at end condition exists when the first character of the I-O status value for the associated file connector is a '1'.

Invalid key condition

The invalid key condition is associated with the I-O status for file connectors that are not associated with a sort file. It exists when the first character of the I-O status value for the associated file connector is a '2'.

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

General format

```
ROUNDED
```

General rules

1. If the arithmetic value cannot be exactly represented in the resultant identifier, the arithmetic value is rounded to the nearest value that can be represented in the resultant identifier. If two such values are equally near, the value whose magnitude is larger is chosen.

SIZE ERROR phrase and size error condition

The size error condition may occur as a result of the execution of an ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT Statement or of the evaluation of an arithmetic expression. Checking of a size error condition may be enabled by specifying the SIZE ERROR phrase of an arithmetic statement.

When the 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. Any exception condition that is raised during item identification for the operands used during the execution of the arithmetic statement is processed as defined for that exception condition and execution of the arithmetic statement ceases. Execution resumes as indicated for that exception condition. If a Size Error condition exists during the execution of the arithmetic statement other than during item identification and a SIZE ERROR phrase is specified for the statement, processing of the size error condition occurs as described below for the SIZE ERROR phrase, and no Size Error exception declaratives are performed.

The size error condition exists in the following cases:

1. if the rules for evaluation of exponentiation are violated;
2. if the divisor in a divide operation or in a DIVIDE Statement is zero;
3. if, after radix point alignment and any rounding specified by the ROUNDED phrase, the absolute value of the result of an arithmetic statement exceeds the largest value that may be contained in the associated resultant data item;
4. if an arithmetic operation on the intermediate data item would cause the new value to be outside of the allowed range;
5. if the rules for a statement or an expression explicitly specify that a Size Error exception condition is set to exist.

If the size error condition exists and the SIZE ERROR phrase is specified, the following occurs:

1. If the size error condition occurred 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 did not occur are the same as they would have been if the size error condition had not existed 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 rule 2, 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 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 SIZE ERROR phrase, control is transferred to the end of the arithmetic statement and the NOT SIZE ERROR phrase, if specified, is ignored.

If the size error condition exists and a SIZE ERROR phrase is not specified, an exception is raised in the following cases:

1. if the rules for evaluation of exponentiation are violated,
2. if the divisor in a divide operation or the DIVIDE Statement is zero,
3. if the absolute value of the result of an arithmetic statement exceeds the largest value that may be contained in the associated resultant data item
4. if an arithmetic operation on a standard intermediate data item would cause the new value to be outside of the allowed range,

and processing proceeds as specified in "Fatal exception conditions" above. If a NOT SIZE ERROR phrase is specified, it is ignored.

If no size error condition occurs during the execution of the arithmetic operations specified by an arithmetic statement or expression 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 expression 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 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 SIZE ERROR phrase, control is transferred to the end of the arithmetic statement.

CORRESPONDING phrase

For the purpose of this discussion, D1 and D2 are identifiers that refer to alphanumeric group items, national group items, or variable-length groups.

NOTE - When D1 and D2 refer to national groups, D1 and D2 are processed as group items and not as elementary items.

A pair of data items correspond if:

1. A data item in D1 and a data item in D2 are not implicitly or explicitly described with the keyword 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. Neither data item contains an OCCURS, REDEFINES, or RENAMES clause or is of class index, object, pointer or handle.
4. Neither data item is subordinate to a group item that is subordinate to D1 or D2 when the group item contains an OCCURS or REDEFINES clause.
5. 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.

For the arithmetic statements with the CORRESPONDING phrase, if the SIZE ERROR phrase is not specified and any of the implied statements raises a size error condition, the Size Error exception condition for the last of the implied statements that raised a size error condition is set to exist after all of the implied statements are completed.

For any statement with the CORRESPONDING phrase, if any of the implied statements would raise the Data Incompatible exception condition to exist, the Data Incompatible exception condition is set to exist after all of the implied statements are completed.

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. For ADD, DIVIDE, MULTIPLY, and SUBTRACT Statements when native arithmetic is in effect:

   a. When none of the operands is an intrinsic function or a data item described with usage Double, Float, Signed-Int, Integer, Signed-Long, Signed-Short, Unsigned-Int, Unsigned-Long or Unsigned-Short, the composite of operands shall not contain more than 18 digits.

   b. When any of the operands is an intrinsic function or a data item described with usage Double, Float, Signed-Int, Integer, Signed-Long, Signed-Short, Unsigned-Int, Unsigned-Long or Unsigned-Short, the composite of the operands that are not intrinsic functions or data items described with usage Double, Float, Signed-Int, Integer, Signed-Long, Signed-Short, Unsigned-Int, Unsigned-Long or Unsigned-Short shall not contain more than 18 digits. The composite of operands is a hypothetical data item resulting from the superimposition of specified operands in a statement aligned on their decimal points.

3. When standard, standard-binary or standard-decimal arithmetic is in effect, each arithmetic statement is defined in terms of one or more arithmetic expressions. Storing of a standard intermediate data item in a resultant-identifier shall be according to the rules for the MOVE Statement. The ROUNDED phrase applies only to this move.

   NOTE - When standard arithmetic is in effect, a size error condition may occur during the execution of arithmetic operations used to compute the result as well as during the final move.

4. The arithmetic statements may have single or multiple resultant identifiers. These statements have common rules for storing in the resultant identifiers. The execution of these statements proceeds in the following order:

   a. 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 are 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. If the size error condition is raised during the initial evaluation, none of the resultant data items are changed and execution proceeds as indicated in the SIZE ERROR phrase above and in the size error condition.

   b. If the size error condition was not raised during the initial evaluation, 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 a. If the size error condition is raised when attempting to store in a resulting data item, only that data item remains unchanged and processing proceeds to the next resulting data item to the right

When resolving addition and subtraction operations with decimal values, if all operands are equal or smaller than 18 digits in size, the result is internally stored into a Java long item, even if it would require more space. When one or more of the operands are greater than 18 digits in size, instead, the result is internally stored into a internal object that is slower than Java long but provides better precision. Due to this rule, if the program needs to handle numbers over 18 digits in size, it's strongly suggested to make the involved numeric variables greater than 18 digits in size, or the results may be incorrect in the decimal part.

The following code causes incorrect result:

```
01 W1 PIC 9(18).
01 W2 PIC 9(4)v9(4).
01 W3 PIC 9(18)v9(4).

MOVE 111111111111111111 TO W1.
MOVE 3333.5555 TO W2.
ADD W1 TO W2 GIVING W3.
```

In order to obtain the correct result, W1 should be defined greater than 18 digits, for example:

```
01 W1 PIC 9(19).
```

or, alternatively, you should compile the program with the -cfp36 option.

THROUGH phrase

This specification applies to THROUGH phrases specified in the VALUE clause and the EVALUATE Statement.

A THROUGH phrase specifies a range of values, *literal-1* through *literal-2*. The set of values included in the range is determined by the following rules:

1. When the range of values is defined by numeric literals, the range of values includes *literal-1*, *literal-2*, and all algebraic values between *literal-1* and *literal-2*.
2. When the range of values is defined by alphanumeric or national literals, the range of values depends on the collating sequence used for evaluation of the range.

   When the value of *literal-1* is greater than the value of *literal-2* in the collating sequence in effect at runtime, the Invalid Range exception condition is set to exist, and, upon completion of any exception processing, execution proceeds as if the range of values were empty.

Conformance for parameters and returning items

The conformance rules for parameters and returning items apply at compile time when an explicit reference is made to them from a syntax rule.

NOTE - Conformance rules for parameters and returning items are checked at compile time for:

- an INVOKE Statement on an object reference that is not a universal object reference,
- a program-prototype format of the CALL Statement.

The conformance rules for parameters and returning items apply at runtime when an explicit reference is made to them from a general rule.

NOTE - Conformance rules for parameters and returning items are checked at runtime for:

- a universal object reference
- the on-overflow and on-exception formats of the CALL Statement

NOTE - Conformance rules for parameters and returning items are checked for an object-view at compile time. Additional conformance rules for the object referenced by the object view are given in the rules of object-view; these are checked at runtime.

Parameters

The number of arguments in the activating element shall be equal to the number of formal parameters in the activated element, with the exception of trailing formal parameters that are omitted from the list of arguments of the activating element.

If both an argument and its corresponding formal parameters are elementary items, the conformance rules for elementary items apply; otherwise, the conformance rules for group items apply.

NOTE - A national group is treated as an elementary item.

Group items

If either the formal parameter or the argument is an alphanumeric group item:

1. If the argument is passed by reference, that argument or the formal parameter corresponding to that argument shall be an alphanumeric 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 bytes as the corresponding argument.
2. If the argument is passed by content, the conformance rules are the same as for a MOVE Statement with the argument as the sending operand and the corresponding formal parameter as the receiving operand.

   NOTE - If an argument is a group with a level number other than 1 and its subordinate items are described such that the implementation inserts slack bits or bytes, the alignment of the subordinate elementary items might not correspond between the argument and the formal parameter.

For an argument or formal parameter that is described as an occurs-depending group item passed by reference, the maximum length is used. For an occurs-depending group item passed by content, the length of the argument is determined by the rules of the `OCCURS` clause for a sending data item.

Elementary items

The conformance rules for elementary items depend on whether the argument is passed by reference or by content.

Elementary items passed by reference

If either the formal parameter or the corresponding argument is an object reference, the corresponding argument or formal parameter shall be an object reference following these rules:

1. If either the argument or the formal parameter is a universal object reference, the corresponding formal parameter or argument shall be a universal object reference.
2. If either the argument or the formal parameter is described with an interface-name, the corresponding formal parameter or argument shall be described with the same interface-name.
3. If either the argument or the formal parameter is described with a class-name, the corresponding formal parameter or argument shall be described with the same class-name, and the `FACTORY` phrases shall be the same.

If either the argument or the formal parameter is of class pointer, the corresponding formal parameter or argument shall be of class pointer and the corresponding items shall be of the same category. If either is a restricted pointer, both shall be restricted and of the same type.

If neither the formal parameter nor the argument is of class object or pointer, the conformance rules are the following:

1. If the activated element is a program for which there is no program-specifier in the `REPOSITORY` paragraph of the activating element and there is no `NESTED` phrase specified on the CALL Statement, the formal parameter shall be of the same length as the corresponding argument.
2. If the activated element is a method, then the definition of the formal parameter and the definition of the argument shall have the same `BLANK WHEN ZERO`, `JUSTIFIED`, `PICTURE`, `USAGE`, and `SIGN` clauses, with the following exceptions:

   a. Currency symbols match if and only if the corresponding currency strings are the same.

b. Period picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both the activating and the activated runtime elements or for neither of them. Comma picture symbols match if and only if the DECIMAL-POINT IS COMMA clause is in effect for both the activating and the activated runtime elements or for neither of them.

Additionally:

a. A national group item matches an elementary data item of usage national described with the same number of national character positions.
b. If the formal parameter is described with the ANY LENGTH clause, its length is considered to match the length of the corresponding argument.
c. If the argument is described with the ANY LENGTH clause, the corresponding formal parameter shall be described with the ANY LENGTH clause.

Elementary items passed by content

If the formal parameter is of class pointer or an object reference, the conformance rules shall be the same as if a SET Statement were performed in the activating runtime element with the argument as the sending operand and the corresponding formal parameter as the receiving operand.

If the formal parameter is not of class object or pointer, the conformance rules are the following:

1. If the activated element is a program, the formal parameter shall be of the same length as the corresponding argument.
2. If the activated element is a method, then the conformance rules depend on the type of the formal parameter as specified in the following rules:

   a. If the formal parameter is numeric, the conformance rules are the same as for a COMPUTE Statement with the argument as the sending operand and the corresponding formal parameter as the receiving operand.
   b. If the formal parameter is an index data item, the conformance rules are the same as for a SET Statement with the argument as the sending operand and the corresponding formal parameter as the receiving operand.
   c. If the formal parameter is described with the ANY LENGTH clause, its length is considered to match the length of the corresponding argument.
   d. Otherwise, the conformance rules are the same as for a MOVE Statement with the argument as the sending operand and the corresponding formal parameter as the receiving operand

Returning items

A returning item shall be specified in the activating statement if and only if a returning item is specified in the procedure division header of the activated element. A returning item is implicitly specified in the activating element when an inline method invocation is referenced.

The returning item in the activated element is the sending operand, the corresponding returning item in the activating element is the receiving operand.

The rules for conformance between the sending operand and the receiving operand depend on whether at least one of the operands is an alphanumeric group item or both operands are elementary items.

NOTE - A national group is treated as an elementary item.

Group items

If either the sending or the receiving operand is an alphanumeric group item, the corresponding returning item shall be an alphanumeric group item or an elementary item of category alphanumeric, and the receiving operand shall be of the same length as the sending operand.

NOTE - If a returning item in an activating element is a group with a level number other than 1 and its subordinate items are described such that the implementation inserts slack bits or bytes, the alignment of the subordinate elementary items might not correspond between the returning item in the activating runtime element and the returning item in the activated runtime element.

For an operand that is described as a variable-occurrence data item, the maximum length is used.

Elementary items

If either of the operands is an object reference, the corresponding item shall be an object reference, and the conformance rules are the same as if a SET Statement were performed in the activated runtime element with the returning item in the activated element as the sending operand and the corresponding returning item in the activating element as the receiving operand.

If the sending operand is not an object reference, the receiving operand shall have the same `BLANK WHEN ZERO`, `JUSTIFIED`, `PICTURE`, `USAGE`, and `SIGN` clauses, with the following exceptions:

1. Currency symbols match if and only if the corresponding currency strings are the same.
2. Period picture symbols match if and only if the `DECIMAL-POINT IS COMMA` clause is in effect for both the activating and the activated runtime elements or for neither of them.
3. Comma picture symbols match if and only if the `DECIMAL-POINT IS COMMA` clause is in effect for both the activating and the activated runtime elements or for neither of them.

Additionally, if the sending operand is not an object reference:

1. A national group item matches an elementary data item of usage national described with the same number of national character positions.
2. If the receiving operand is described with the `ANY LENGTH` clause, the sending operand shall also be described with the `ANY LENGTH` clause.
3. If the sending operand is described with the `ANY LENGTH` clause, the length of the sending operand is considered to match the length of the receiving operand.

Statement categories

There are four categories of COBOL statements:

- Conditional statements
- Imperative statements
- Delimited scope statements
- Compiler-directing statements

Conditional statements

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value. COBOL statements become conditional when a condition (for example: ON SIZE ERROR, ON OVERFLOW or ON EXCEPTION) is included. The following statements are always considered conditional: IF, EVALUATE, SEARCH.

Imperative statements

An imperative statement either specifies an unconditional action to be taken by the program or is a conditional statement terminated by its explicit scope terminator. COBOL statements used without any clauses and COBOL statements that don't provide clauses (for example: STOP RUN) are considered imperative.

Delimited scope statements

A delimited scope statement is a statement (for example, an IF statement) which is terminated by a matching explicit scope terminator (in this case END-IF). Thus, all the statements between a delimited scope statement and its paired explicit scope terminator are deemed to be contained within that delimited scope statement. Delimited scope statements can be nested, in which case each explicit scope terminator encountered in the program is considered to pair with the nearest preceding unpaired matching delimited scope statement.

Compiler-directing

A compiler-directing statement is a statement that causes the compiler to take a specific action during compilation. isCOBOL supports the following compiler-directing statements: COPY, EJECT, SKIP, REPLACE and USE.

Chapter 6

Procedure Division Statements

Every statement in COBOL is started with a reserved word called a verb. The following section covers the verbs available in isCOBOL.

ACCEPT

Format 1

```
ACCEPT identifier-1 [ FROM mnemonic-name-1 ] [ END-ACCEPT ]
```

Format 2

```
ACCEPT identifier-2 FROM { DATE                        }   [END-ACCEPT]
                         { DAY                         }
                         { CENTURY-DATE                }
                         { DATE YYYYMMDD               }
                         { CENTURY-DAY                 }
                         { DAY YYYYDDD                 }
                         { TIME                        }
                         { DATE-AND-TIME               }
                         { DAY-OF-WEEK                 }
                         { TERMINAL-INFO               }
                         { SYSTEM-INFO                 }
                         { INPUT STATUS                }
                         { ESCAPE KEY                  }
                         { COMMAND-LINE                }
                         { THREAD HANDLE               }
                         { WINDOW HANDLE               }
                         { CONTROL                     }
                         { LINE NUMBER                 }
                         { STANDARD OBJECT Object-Name }
                         { WINDOW OF THREAD Thread-1   }
                         { EXCEPTION STATUS            }
                         { DATE-COMPILED               }
```

Format 3

```
ACCEPT  [ ( {Identifier-3}, {Identifier-4} ) ] {screen-name-1}
          {Integer-3   }  {Integer-4   }
                                              {OMITTED      }

  [ UNTIL Condition-1 ]

  [ Remaining-Phrases ]

  [ MOUSE FLAGS mouse-flags ]

  [ ON {EXCEPTION} [Key-Dest] Imperative-Statement-1 ]
       {ESCAPE   }

  [ NOT ON {EXCEPTION} Imperative-Statement-2 ]
           {ESCAPE   }

  [ END-ACCEPT ]
```

Remaining-Phrases are optional, can appear in any order.

```
  AT Screen-Loc

  AT LINE NUMBER {Identifier-3} [CELL  ]
                 {Integer-1   } [CELLS ]
                                [PIXEL ]
                                [PIXELS]

  AT {COLUMN  } NUMBER {Identifier-4} [CELL  ]
     {COL     }        {Integer-2   } [CELLS ]
     {POSITION}                      [PIXEL ]
     {POS     }                      [PIXELS]

  WITH PROTECTED SIZE Length

  WITH NO ADVANCING

  {ERASE} [TO END OF] {LINE  }
  {BLANK}             {SCREEN}
```

```
{ERASE} [EOS]
{BLANK} [EOL]

WITH [NO] {BELL}
          {BEEP}

{UNDERLINE }
{UNDERLINED}

WITH {BLINKING}
     {BLINK   }

{HIGHLIGHT}
{HIGH     }
{BOLD     }
{LOWLIGHT }
{LOW      }
{STANDARD }

{REVERSE-VIDEO}
{REVERSE      }
{REVERSED     }

SAME

WITH {COLOR } Color-Val
     {COLOUR}

{FOREGROUND-COLOR } IS Fg-Color
{FOREGROUND-COLOUR}

{BACKGROUND-COLOR } IS Bg-Color
{BACKGROUND-COLOUR}

SCROLL [UP  ] [ BY Scrl-Num {LINE } ]
       [DOWN]               {LINES}

OUTPUT {JUSTIFIED}  {LEFT    }
       {JUST     }  {RIGHT   }
                    {CENTERED}

WITH {CONVERSION}
     {CONVERT   }

{ WITH NO ECHO }
{ NO-ECHO      }
{ SECURE       }
{ OFF          }

PROMPT [ CHARACTER IS Prompt-Lit ]

{ DEFAULT [IS Default] }
{ UPDATE               }

ECHO
```

```
  {AUTO          }
  {AUTO-SKIP     }
  {AUTOTERMINATE}
  {TAB           }

  {UPPER}
  {LOWER}

  CURSOR Curs-Offset

  CONTROL Cntrl-String

  {REQUIRED   }
  {EMPTY-CHECK}

  {FULL         }
  {LENGTH-CHECK}

  {ZERO-FILL   }
  {NUMERIC-FILL}

  CONTROL KEY IN Key-Dest

  BEFORE TIME Timeout

  ALLOWING MESSAGES FROM { THREAD Thread-1 }
                         { LAST THREAD     }
                         { ANY THREAD      }
```

Format 4

```
ACCEPT Dest-Item FROM SCREEN

  AT Screen-Loc

  AT LINE NUMBER Line-Num

  AT {COLUMN  } NUMBER Col-Num
     {COL     }
     {POSITION}
     {POS     }

  SIZE Length
[END-ACCEPT]
```

Format 5

```
ACCEPT Dest-Item FROM {CONFIGURATION} Env-Name
                      {ENVIRONMENT  }

  [ ON EXCEPTION Statement-1 ]

  [ NOT ON EXCEPTION Statement-2 ]

  [END-ACCEPT]
```

Format 6

```
ACCEPT {Control-Handle}
       {CONTROL       }

  [Remaining-Phrases]

  [ ON {EXCEPTION} [Key-Dest] Statement-1 ]
       {ESCAPE   }

  [ NOT ON {EXCEPTION} Statement-2 ]
           {ESCAPE   }

  [END-ACCEPT]
```

Remaining-Phrases are optional, can appear in any order.

```
  VALUE IN [MULTIPLE] Value

  AT Screen-Loc

  AT LINE NUMBER Line-Num [CELL  ]
                          [CELLS ]
                          [PIXEL ]
                          [PIXELS]

  AT {COLUMN  } NUMBER Col-Num [CELL  ]
     {COL     }                [CELLS ]
     {POSITION}                [PIXEL ]
     {POS     }                [PIXELS]

  WITH {BELL}
       {BEEP}

  BEFORE TIME Timeout

  CONTROL KEY IN Key-Dest

  ALLOWING MESSAGES FROM { THREAD Thread-1 }
                         { LAST THREAD     }
                         { ANY THREAD      }
```

Format 7

```
ACCEPT EVENT

  BEFORE TIME Timeout

  ALLOWING MESSAGES FROM { THREAD Thread-1 }
                         { LAST THREAD     }
                         { ANY THREAD      }

  [ ON {EXCEPTION} [Key-Dest] Statement-1 ]
       {ESCAPE   }

  [ NOT ON {EXCEPTION} Statement-2 ]
           {ESCAPE   }

  [END-ACCEPT]
```

Format 8

```
ACCEPT value FROM ENVIRONMENT-VALUE

  [ ON EXCEPTION Statement-1 ]

  [ NOT ON EXCEPTION Statement-2 ]

[END-ACCEPT]
```

Syntax Rules

1. Dest-Item is a data item that receives the accepted data.

   If a THREAD HANDLE or WINDOW HANDLE phrase is used, Dest-Item must be a USAGE HANDLE data item of the appropriate type. If the WINDOW HANDLE phrase is used, Dest-Item can also be a PIC X(10) item.
2. Screen-Name is the name of a screen entry declared in the program's Screen Section.
3. Screen-Loc is a numeric literal or data item defined as 4 or 6 digits.
4. Line-Num, Col-Num are numeric data items or literals.
5. Length, Color-Val, Curs-Offset, Timeout, and Scrl-Num are numeric literals or data items.
6. Fg-Color and Bg-Color are integer literals or numeric data items but can not be subscripted.
7. Prompt-Lit is a single-character alphanumeric literal or the figurative constant SPACE, ZERO, or QUOTE.
8. Default is a literal or data item. It references the default entry value.
9. Key-Dest is a numeric data item.
10. Thread-1 is a USAGE HANDLE or HANDLE OF THREAD data item.
11. Statement-1 and Statement-2 are any imperative statements.
12. Condition-1 is any conditional expression.
13. Cntrl-String and Env-Name are nonnumeric literals or data items.
14. *Mnemonic-name-1* can be SYSIN, CONSOLE or CRT. CONSOLE is synonym of SYSIN.
15. Control-Handle must be USAGE HANDLE.
16. Value can be any data item.

17. Code-Dest is defined as a numeric data item
18. The AT phrase precludes the use of the LINE and the COLUMN phrase.
19. The COLOR phrase precludes the use of the FOREGROUND-COLOR and the BACKGROUND-COLOR phrase.
20. The CURSOR phrase may not be specified if a CURSOR phrase is specified in the program's Configuration Section.
21. If the OMITTED option is used, then none of the following phrases may be specified: SIZE, JUSTIFIED, CONVERT, NO ECHO, PROMPT, DEFAULT, ECHO, UPPER, LOWER, REQUIRED, FULL, ZERO-FILL, NUMERIC-FILL or any of the attribute setting phrases (such as COLOR or REVERSE).
22. If the CONTROL KEY phrase is used, the Key-Dest option of the ON EXCEPTION phrase may not be specified.
23. CELL – CELLS and COLUMN – COL and POSITION - POS and PIXEL - PIXELS are synonymous.
24. ERASE and BLANK are synonymous.
25. BELL and BEEP are synonymous.
26. UNDERLINE and UNDERLINED are synonymous.
27. BLINKING and BLINK are synonymous.
28. HIGHLIGHT, HIGH and BOLD are synonymous.
29. LOWLIGHT and LOW are synonymous.
30. REVERSE-VIDEO, REVERSE and REVERSED are synonymous.
31. COLOR and COLOUR are synonymous.
32. FOREGROUND-COLOR and FOREGROUND-COLOUR are synonymous.
33. BACKGROUND-COLOR and BACKGROUND-COLOUR are synonymous.
34. LINE and LINES, in the SCROLL phrase, are synonymous.
35. JUSTIFIED and JUST are synonymous.
36. CONVERSION and CONVERT are synonymous.
37. AUTO, AUTO-SKIP and AUTOTERMINATE are synonymous.
38. REQUIRED and EMPTY-CHECK are synonymous.
39. FULL and LENGTH-CHECK are synonymous.
40. ZERO-FILL and NUMERIC-FILL are synonymous.
41. CONFIGURATION and ENVIRONMENT are synonymous.
42. ON ESCAPE and ON EXCEPTION (where ESCAPE is allowed) are synonymous.
43. Object-Name is an alphanumeric literal or data item.
44. ON EXCEPTION in Format 8 is not allowed under -ca option.

General Rules

1. An ACCEPT Statement can return one of the following pre-defined exception values.

91 W-TERMINATE	ACCEPT has been terminated by another thread through STOP THREAD statement
95 W-MESSAGE	message received during ACCEPT with ALLOWING MESSAGES clause
96 W-EVENT	an event has been raised during the ACCEPT of a graphical screen
97 W-NO-FIELDS	no input fields are available

98 W-CONVERSION-ERROR	error during numeric conversion
99 W-TIMEOUT	time expired during ACCEPT with BEFORE TIME clause

Format 1

2. 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.
3. 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.
4. 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. If identifier-1 references a zero-length item, all the characters of the transferred data are ignored.

Format 2

5. 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, and TIME reference the current date 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 source unit.
6. DATE without the phrase YYYYMMDD behaves as if it had been described as an unsigned elementary integer data item of usage display six 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.
3-4	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.

7. DATE with the phrase YYYYMMDD and CENTURY-DATE behave as if they have 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 Positions	Contents
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.

8. DAY without the phrase YYYYDDD 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.

9. DAY with the phrase YYYYDDD and CENTURY-DAY behave as if they 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
1-4	Four numeric characters of the year in the Gregorian calendar.
5-7	Three numeric characters of the day of the year in the range 001 through 366.

10. 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 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.

11. DATE-AND-TIME behaves as if it had been described as an unsigned elementary integer data item of usage display sixteen digits in length, the character positions of which, numbered from left to right, are:

Character Positions	Contents
1-4	Four numeric characters of the year.
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.
9-10	Two numeric characters of the hours past midnight in the range 00 through 23.
11-12	Two numeric characters of the minutes past the hour in the range 00 through 59.
13-14	Two numeric characters of the seconds past the minute in the range 00 through 59
15-16	Two numeric characters of the hundredths of a second past the second in the range 00 through 99.

12. DAY-OF-WEEK behaves as if it had been described as an unsigned elementary numeric integer data item one digit in length and of usage display. In DAY-OF-WEEK, the value 1 represents Monday, 2 represents Tuesday, 3 represents Wednesday, ... , 7 represents Sunday.
13. The TERMINAL-INFO option causes information about the user's terminal to be moved to identifier-2. This information is returned in the following format, as contained in the TERMINAL-ABILITIES group item defined in iscobol.def:

```
01  terminal-abilities.
    03  terminal-name                      pic x(10).
    03  filler                             pic x.
        88  has-reverse                    value "Y".
    03  filler                             pic x.
        88  has-blink                      value "Y".
    03  filler                             pic x.
        88  has-underline                  value "Y".
    03  filler                             pic x.
        88  has-dual-intensity             value "Y".
    03  filler                             pic x.
        88  has-132-column-mode            value "Y".
    03  filler                             pic x.
        88  has-color                      value "Y".
    03  filler                             pic x.
        88  has-line-drawing               value "Y".
    03  number-of-screen-lines             pic 9(3).
    03  number-of-screen-columns           pic 9(3).
    03  filler                             pic x.
        88  has-local-printer              value "Y".
    03  filler                             pic x.
        88  has-visible-attributes         value "Y".
    03  filler                             pic x.
        88  has-graphical-interface        value "Y".
    03  usable-screen-height               pic x(2) comp-x.
    03  usable-screen-width                pic x(2) comp-x.
    03  physical-screen-height             pic x(2) comp-x.
    03  physical-screen-width              pic x(2) comp-x.
    03  filler                             pic x.
        88  is-remote                      value "Y".
    03  client-machine-name                pic x(64).
    03  filler                             pic x.
    03  client-user-id                     pic x(20).
```

Fields have the following meaning:

Field	Information
terminal-name	It returns “xterm”. The value can be customized via the configuration property iscobol.terminal.info.name

Field	Information
has-reverse has-blink has-underline has-dual-intensity has-132-column-mode has-color has-line-drawing has-local-printer has-visible-attributes has-graphical-interface	These flags are set to "Y" if the terminal has the corresponding ability or to "N" otherwise. The values can be customized via the configuration properties iscobol.terminal.info.reverse iscobol.terminal.info.blink iscobol.terminal.info.underline iscobol.terminal.info.dual_intensity iscobol.terminal.info.132column iscobol.terminal.info.color iscobol.terminal.info.drawing iscobol.terminal.info.printer iscobol.terminal.info.attributes iscobol.terminal.info.graphic
number-of-screen-lines number-of-screen-columns	These items hold the number of whole lines and columns (respectively) in the current window. If no window has been created, these values are set to the size of the default application window. The values can be customized via the configuration properties iscobol.terminal.info.screen.lines iscobol.terminal.info.screen.columns
usable-screen-height usable-screen-widtht	These items hold the height and width (respecitively) of the usable portion of the user's display device in pixels. The values can be customized via the configuration properties iscobol.terminal.info.screen.usable.height iscobol.terminal.info.screen.usable.width
physical-screen-height physical-screen-widtht	These items are the same as usable-screen-height and usable-screen-width except that they include the entire screen instead of just the usable portion of the screen. The values can be customized via the configuration properties iscobol.terminal.info.screen.physical.height iscobol.terminal.info.screen.physical.width
is-remote	This flag is set to "Y" if the program is running in thin client mode, or to "N" if the program is running in stand-alone mode.
client-machine-name	This item is set when running in thin client mode and it holds the machine name of the client PC.
client-user-id	This item is set when running in thin client mode and it holds the user name of the client PC.

For performance reasons, the runtime only obtains the screen width and screen height information the first time that TERMINAL-INFO is accepted from the TERMINAL-ABILITIES. The next times the runtime returns the stored information. Due to this rule, the returned information becomes unreliable if the user changes the screen resolution in the system during the runtime session. In order to disable the optimization and have the runtime inquire the system every time that TERMINAL-INFO is accepted from TERMINAL-ABILITIES, set iscobol.terminal.info.refresh_monitor (boolean) * to true in the configuration.

14. The SYSTEM-INFO option causes some general information about the runtime to be moved to identifier-2. This information is returned in the following format, as contained in the SYSTEM-INFORMATION group item

defined in iscobol.def:

```
01  system-information.
    03  operating-system                        pic x(10).
        88  os-is-msdos                         value "ms-dos".
        88  os-is-os2                           value "OS/2".
        88  os-is-vms                           values "vms", "vax/vms".
        88  os-is-unix                          value "Linux", "AIX", "HP-
UX", "SunOS", "Solaris".
        88  os-is-linux                         value "Linux".
        88  os-is-aos                           value "aos/vs".
        88  os-is-
windows                        values "Windows 95", "Windows 98", "Windows Me".
        88  os-is-win-
nt                          values "Windows NT", "Windows 20", "Windows XP", "Windows V
i",
                                                        "Windows 7", "WINDOWS", "Windo
ws Se", "Windows 8", "Windows 8.",
                                                        "Windows 10", "Windows 11".
        88  os-is-win-
family                      values "Windows NT", "Windows 95", "Windows 98", "Windows M
e",
                                                        "Windows 20", "Windows XP", "W
indows Vi", "WINDOWS", "Windows 7",
                                                        "Windows Se", "Windows 8", "Wi
ndows 8.", "Windows 10", "Windows 11".
        88  os-is-amos                          value "amos".
        88  os-is-mpe                           value "MPE/iX".
        88  os-is-mpeix                         value "MPE/iX".
        88  os-is-mac                           value "Mac OS", "Mac OS X".
    03  user-id                                 pic x(12).
    03  station-id                              pic x(12).
    03  filler                                  pic x.
        88  has-indexed-read-previous           value "Y".
    03  filler                                  pic x.
        88  has-relative-read-previous          value "Y".
    03  filler                                  pic x.
        88  can-test-input-status               value "Y".
    03  filler                                  pic x.
        88  is-multi-tasking                    value "Y".
    03  runtime-version.
        05  runtime-major-version               pic 99.
        05  runtime-minor-version               pic 99.
        05  runtime-release                     pic 99.
    03  filler                                  pic x.
        88  is-plugin                           value "Y".
    03  serial-number                           pic x(20).
    03  filler                                  pic x.
        88  has-large-file-support              value "Y".
    03  filler                                  pic x.
    03  filler                                  pic x.
        88  is-64-bit                           value "Y".
```

Fields have the following meaning:

Field	Information
operating-system	This field holds the name of the operating system. The name may be truncated due to the field picture. It's good practice to reference the corresponding 88 level items in the source. The value can be customized via the configuration property iscobol.os.name.
user-id	This field holds the system user name. The value can be customized via the configuration property iscobol.user.name.
station-id	This field is set to spaces by default. The value can be customized via the configuration property iscobol.station
has-indexed-read-previous has-relative-read-previous can-test-input-status is-multitasking has-large-file-support	These flags are always set to "Y".
runtime-version	This field holds the isCOBOL runtime version and release number.
is-plugin	This flag is always set to "N".
serial-number	This field returns the serial number associated to the runtime license.
is-64-bit	This flag is set to "Y" if the JVM bitness is 64 bit, else it's set to "N".

Note - If your current JVM was released before the current operating system, then the *operating-system* field value may not be accurate.

15. The INPUT STATUS form of the ACCEPT statement returns a value that indicates whether or not there is data currently available from the standard input. Identifier-2 should be described as PICTURE 9(1). It receives 1 if input is currently available, 0 if not.
16. The ESCAPE option of the ACCEPT statement returns the last termination key value.
17. The ACCEPT FROM COMMAND-LINE option causes the contents of the original command line to be moved to identifier-2. It returns elements that appear after the program name as a single string.
18. The ACCEPT FROM THREAD HANDLE statement moves the thread ID of the executing thread to identifier-2.
19. The WINDOW HANDLE option causes identifier-2 to receive the handle of the current window.
20. CONTROL option causes identifier-2 to receive the handle of the current control.
21. The ACCEPT FROM LINE NUMBER statement has no effect. The syntax is supported for compatibility and is treated as commentary.
22. The ACCEPT FROM STANDARD OBJECT returns a handle to one of the system's pre-defined resources. The resource returned depends on the value of Object-Name. Possible values are:

    - Default-Font
    - Fixed-Font
    - Large-Font
    - Lm-Resize
    - Lm-Scale

- Medium-Font
- Small-Font
- Traditional-Font

23. The ACCEPT FROM WINDOW OF THREAD returns the handle of the active window of a given thread.
24. The ACCEPT FROM EXCEPTION STATUS is supported for documentation purposes only and it corresponds to MOVE ZERO TO identifier-2.
25. The ACCEPT FROM DATE-COMPILED returns the data elements year, month and day for the date the program compilation started. It is a constant for any particular compilation. The sequence is YYYYMMDD.

Format 3

26. The ACCEPT statement accepts data entered by the user. The data accepted is placed in Dest-Item. If the OMITTED option is used instead of Dest-Item, the user must supply a termination key, e.g., a function key to end the entry with no data transferred.
27. The AUTO phrase causes a field to terminate as soon as it is filled with data. The TAB phrase requires a termination key to finish entry.
28. 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 screen 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 that 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

29. The LINE and COLUMN phrases give the position on the 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.
    The line and column coordinates can be specified between parenthesis soon after the ACCEPT word. This is a Microsoft COBOL extension and requires the -cms compile flag.
30. If the LINE phrase is not specified, the screen record starts on line 1.
31. If the COLUMN phrase is not specified, the screen record starts in column 1.
32. The SIZE phrase specifies the number of screen positions to return. If the SIZE phrase is missing, then the size of dest-item is used. If the SIZE phrase specifies fewer characters than the size of dest-item, then dest-item is space-filled on the right. Double-byte characters count for 2, for example, an ACCEPT with SIZE 10 allows you

to input 10 Latin characters or 5 Chinese characters.

33. 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 may be modified as the operator keys data.
34. Data entered by the operator in the current screen item shall be consistent with the PICTURE clause of that item. If the screen item is numeric, the entered data shall be acceptable as an argument to the NUMVAL function. If the screen item is numeric-edited, the entered data shall be acceptable as an argument to the NUMVAL-C function.
35. 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 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:

    A. If the screen item is numeric, the data is transferred as though the following statement were executed:

        COMPUTE receiving-field = FUNCTION NUMVAL (screen-item)

    B. If the screen item is numeric-edited, the data is transferred as though the following statement were executed:

        COMPUTE receiving-field = FUNCTION NUMVAL-C (screen-item)

    C. Otherwise, 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.
36. If the CURSOR clause is specified in the special-names paragraph, 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.
37. The CONTROL phrase provides the ability to modify the static attributes of the ACCEPT statement at runtime. The CONTROL data item is treated as a series of comma-separated keywords that control the action of the statement. Within the CONTROL data item, spaces are ignored and lower-case letters are treated as if they

were upper-case. The keywords allowed in cntrl-string are:

```
ERASE, ERASE EOL, ERASE EOS, NO ERASE
BEEP, NO BEEP
HIGH, LOW, STANDARD, OFF
BLINK, NO BLINK
REVERSE, NO REVERSE
TAB, NO TAB
PROMPT, NO PROMPT
CONVERT, NO CONVERT
UPDATE, NO UPDATE
ECHO, NO ECHO
UPPER, NO UPPER, LOWER, NO LOWER
UNDERLINED, NO UNDERLINE
LEFT, RIGHT, CENTERED, NO JUST
SAME
FCOLOR
BCOLOR
"GRAPHICS"
```

Any other keywords and spaces are discarded. If more than one keyword from within the above lines appears in cntrl-string, then only the rightmost one in the data item is used. Each of the keywords performs the same action as the statically declared attribute of the same name. When a CONTROL item conflicts with the statically declared attributes of the ACCEPT statement, the actions specified in the CONTROL item take precedence. The FCOLOR and BCOLOR keywords are used to set foreground and background colors respectively. These keywords must be followed by an equals sign and the name of a color taken from the following list: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, and WHITE. The named color becomes the default foreground or background color for the window. Note that this is different from the COLOR phrase, which sets the color only for the current ACCEPT statement. The FCOLOR and BCOLOR keywords set the default colors for every subsequent ACCEPT until explicitly changed. "GRAPHICS" allows you to create single line boxes.

38. If the execution of the ACCEPT statement results in a successful completion with normal termination, the ON EXCEPTION phrase, if specified, is ignored and control is transferred to the end of the ACCEPT 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 ACCEPT statement.
39. If the execution of the ACCEPT statement results in an unsuccessful completion and is terminated by a function key stroke, then:
    A. If the ON EXCEPTION phrase is specified in the ACCEPT statement, control is transferred to imperative-statement-1. Execution then continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement 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 ACCEPT statement and the NOT ON EXCEPTION phrase, if specified, is ignored.
    B. If the ON EXCEPTION phrase is not specified in the ACCEPT statement, control is transferred to the end of the ACCEPT statement and the NOT ON EXCEPTION phrase, if specified, is ignored.
40. The MOUSE FLAGS option triggers the termination of the accept when a mouse exception occurs.

The value of *mouse-flags* is the sum between one or more of the following values:

Mouse Action	Value
Allow left button down	2
Allow left button up	4
Allow left button double click	8
Allow middle button down	16
Allow middle button up	32
Allow middle button double click	64
Allow right button down	128
Allow right button up	256
Allow right button double click	512
Allow mouse move	1024

For example, in order to intercept all mouse actions, use

```
MOUSE FLAGS 2046
```

The following crt status values are returned by mouse actions:

Mouse Action	Crt Status Value
Mouse moved	80
Left button pushed	81
Left button released	82
Left button double-clicked	83
Middle button pushed	84
Middle button released	85
Middle button double-clicked	86
Right button pushed	87
Right button released	88
Right button double-clicked	89

In the general model for graphical user interfaces, the system directs events to the window where the event occurred. This window owns the event. The effect of this is most noticeable when you examine what happens when controls interact with the mouse. As the mouse moves across the application screen, the various windows that the pointer passes over each receive the appropriate events. If you look at an application

screen that has several controls, the application window receives those mouse events that occur when the pointer is in the application window, but not over any of the controls. When the mouse pointer is over a control, that control receives the mouse events. This means that MOUSE FLAGS affects the behavior of the mouse only when it is not over a control. When the mouse is over a control, that control does its own mouse processing.

Format 4

41. A Format 4 ACCEPT is performed on the last displayed window.
42. If POSITION is set to zero, then the ACCEPT of dest-item starts immediately after the last DISPLAY, as if POSITION was omitted.

Additional phrases, CURSOR, CONTROL KEY, BEFORE TIME, ON EXCEPTION are described below:

CURSOR Phrase

43. The CURSOR phrase specifies the initial cursor offset (1-based), from the beginning of the field. The leftmost position of the ACCEPT field is offset 1. If the CURSOR phrase is omitted or zero, then an offset of 1 is used.

CONTROL KEY Phrase

44. Some ACCEPT statements terminate automatically when the input field is filled. When this occurs, the termination key value is the value (a decimal number) defined by the AUTO-RETURN keyword. This value is returned in the CONTROL KEY clause of the ACCEPT statement. The default value is zero.
45. Key-Dest is a numeric item that receives the value of the key that terminated input. The keys allowed by the CONTROL KEY phrase and their returned values are defined in the Keyboard Configuration section of the isCOBOL UserGuide.

BEFORE TIME Phrase

46. The BEFORE TIME phrase allows you to automatically terminate an ACCEPT statement after a certain amount of time has passed. The timeout value specifies the time to wait in hundredths of a second. For example, "BEFORE TIME 500" specifies a timer value of 5 seconds.
47. The user must enter data to the ACCEPT statement before the timer elapses. As soon as the user starts entering data, the timer is canceled and the user may take as much time as desired to complete the entry. If the user does not enter any data before the timer elapses, then the ACCEPT statement terminates and returns a value exactly as if the user had typed the "enter" key. An exception condition is then raised and the exception key value is set to "99"
48. The BEFORE TIME phrase affects only ACCEPT statements performed on a window.

ALLOWING MESSAGES Phrase

49. The ALLOWING MESSAGES phrase causes the ACCEPT statement to terminate when a message is sent from the appropriate thread, as follows:
    A. The THREAD Thread-1 option allows messages from the thread identified by Thread-1.
    B. The LAST THREAD option allows for messages from the "last" thread.
    C. The ANY THREAD option allows messages from any thread.
50. If an allowed message is available when the ACCEPT begins, or one arrives while the ACCEPT is active, the ACCEPT terminates with an exception value of "95". The exception occurs even if exceptions are not otherwise allowed in the ACCEPT.
51. When an ACCEPT terminates because of a message, the intended data item is not updated. This affects the following cases:
    A. In Format 3, Dest-Item is not updated.
    B. In Format 6, Value is not updated.
52. An ACCEPT statement that is suspended (because another thread has an active ACCEPT) terminates when an

allowed message arrives.

53. If the ALLOWING MESSAGES phrase is omitted, messages will not terminate the ACCEPT. Instead, they are queued as normal

ON EXCEPTION Phrase

54. When this phrase is used, Statement-1 is executed when an exception condition occurs. An exception condition occurs under the following circumstances:
    A. An end-of-file condition occurs on the console.
    B. A BEFORE TIME phrase was specified and the ACCEPT statement timed out.
    C. An exception key was used to terminate input.
    D. A conversion error occurred when numeric data was entered with the CONVERSION phrase.
    E. A Message terminates the ACCEPT.
    F. An Event terminates the ACCEPT.
    G. The screen contains no input fields, or all input fields are protected or disabled.
55. Key-Dest, if specified, causes this phrase to have all of the effects of the CONTROL KEY phrase in addition to its normal effects.
56. If you specify an ON EXCEPTION phrase, then exception keys will be allowed for the ACCEPT statement. Otherwise exception keys will be disabled unless you use the CONTROL KEY phrase.
57. If the NEXT SENTENCE option is used, then control will pass to the next executable sentence when an exception condition exists.
58. If the NOT ON EXCEPTION phrase is specified, then Statement-2 executes if no exception condition exists.

Format 5

59. A Format 5 ACCEPT statement returns values from the user's environment or the isCOBOL Framework configuration properties. Env-Name is the name of the environment setting whose value is to be returned. If the literal name of this item is used, then it must enclosed in quotes. The value returned from this item is moved to dest-item.
60. The runtime will normalize Env-Name by making it lower-case and translating hyphens to underscores, then it will search for Env-Name according to the rules described in Configuration.
    If an entry is found, then its value is moved to dest-item.
    If no matching entry is found, or if the Env-Name is the name of a configuration variable whose value cannot be returned, spaces are moved to dest-item and Statement-1, if specified, is executed.
    If a legal matching entry is found, then Statement-2 (if specified) is executed.

Format 6

61. The Format 6 ACCEPT activates the control identified by control-handle. The user interacts with the control until some terminating event occurs. The event that caused the termination is then stored in Key-Dest, and the control's current value is stored in Value. Then the ACCEPT statement terminates.
62. If the CONTROL phrase is used, the runtime activates the control located at the screen position specified by the AT, LINE, and COLUMN phrases in the current window. The runtime maintains a list of controls in each window. When attempting to activate a control at a specific location, the runtime searches this list, using the first control it finds that exactly matches the given location. The list is maintained in the order in which the controls are created. If the runtime does not find a control at the specified location, it returns an exception value of "96" (the same as doing an ACCEPT of a invalid control handle).
63. Key-dest names a data item that will receive a code indicating the terminating event. The program's CRT STATUS (if any) also receives the termination code.
64. When the ACCEPT statement terminates, the current value of the control is moved to value in accordance with the rules for the MOVE statement. The type of control determines the source format of the value.

65. The BEFORE TIME, BELL, CONTROL KEY and ALLOWING MESSAGES phrases operate in the same manner as they do in a Format 3 ACCEPT.

Format 7

66. The runtime waits for a terminating event to occur. The event that caused the termination is stored in Key-Dest. Then the ACCEPT statement terminates.
67. Key-Dest names a data item that will receive a code indicating the terminating event. The program's CRT STATUS (if any) also receives the termination code.
68. The BEFORE TIME and ALLOWING MESSAGES phrases operate in the same manner as they do in a Format 3 ACCEPT.

Format 8

69. A Format 8 ACCEPT fetches the value of an environment or configuration variable stored with a Format 12 DISPLAY statement.
70. The value data item should be of a size and type that allow the value of the environment or configuration variable to be accommodated.

Examples

Format 1

```
accept ws-code from console
```

Format 2 - Several examples

```
*> Accept current date
accept my-date from date
*> sample value of my-date : 131220

*> Accept current date with 4 digit year
accept my-date from date yyyymmdd
*> sample value of my-date : 20131220

*> Accept current day of the year
accept myday from day yyyyddd
*> sample value for myday : 2013335

*> Accept current day and time
accept mydatetime from date-and-time
*> sample value for mydatetime : 2013121910593228

*> Accept parameters from command line
accept all-param from command-line
*> See unstring sample to put every parameter word on a separate variable
```

Format 3 - Display and Accept a screen defined in Screen Section

```
       working-storage section.
       01 window-handle usage handle.
       01 cust-values.
          05 ws-cust-code  pic x(5).
          05 ws-cust-name  pic x(50).

       screen section.
       01 screen-1.
          03 scr-cust-code Entry-Field
             using ws-cust-code
             line 3.0
             column 21.4
             size 12.7 cells
             lines 3.8 cells
             id 1
             3-d.
          03 scr-cust-name Entry-Field
             using ws-cust-name
             line 8.7
             column 21.4
             size 24.5 cells
             lines 4.6 cells
             id 2
             3-d.
          03 scr-lab-1 Label
             line 2.7
             column 4.5
             size 13.5 cells
             lines 3.7 cells
             id 3
             title "Code :".
          03 scr-lab-2 Label
             line 9.3
             column 4.5
             size 13.5 cells
             lines 3.7 cells
             id 4
             title "Name :".
          03 scr-pb-save Push-Button
             line 16.4
             column 15.7
             size 15.5 cells
             lines 6.4 cells
             id 5
             title "Save".
         ...
```

```
      procedure division.
      display-and-accept.
         display standard  window background-low
             screen line 41
             screen column 91
             size 49.7
             lines 24.9
             cell width 10
             cell height 10
             label-offset 20
             color 257
             modeless
             title "Customers"
             handle window-handle.
          display screen-1.
          perform until exit-pushed
             accept screen-1 on exception
                perform is-screen-1-evaluate-func
             end-accept
          end-perform.
          destroy window-handle.
      ...
```

Format 4 - Get the text displayed on specific position of a character-based screen

```
display "Customer Name : " line 5 col 10
display cust-name line 5 col 30
...
accept tmp-cust-name from screen line 5 col 30
```

Format 5 - Get some configuration properties and environment variable values

```
*> Get the TEMP environment variable value
accept mytemp-dir from environment "temp"
*> Get the iscobol.file.index property value
accept my-file-index from environment "file.index"
*> Get the value of a custom property defined as:
*> iscobol.mycustom.location=Location01
accept my-location from environment "mycustom.location"
```

Format 6 - Display screen defined in screen section, accept only one control

```
       working-storage section.
       01 window-handle usage handle.
       01 cust-values.
          05 ws-cust-code  pic x(5).
          05 ws-cust-name  pic x(50).

       screen section.
       01 screen-1.
          03 scr-cust-code Entry-Field
             using ws-cust-code
             line 3.0
             column 21.4
             size 12.7 cells
             lines 3.8 cells
             id 1
             3-d.
          03 scr-cust-name Entry-Field
             using ws-cust-name
             line 8.7
             column 21.4
             size 24.5 cells
             lines 4.6 cells
             id 2
             3-d.
          03 scr-lab-1 Label
             line 2.7
             column 4.5
             size 13.5 cells
             lines 3.7 cells
             id 3
             title "Code :".
          03 scr-lab-2 Label
             line 9.3
             column 4.5
             size 13.5 cells
             lines 3.7 cells
             id 4
             title "Name :".
          03 scr-pb-save Push-Button
             line 16.4
             column 15.7
             size 15.5 cells
             lines 6.4 cells
             id 5
             title "Save".
         ...
```

```
      procedure division.
      display-and-accept.
         display standard  window background-low
             screen line 41
             screen column 91
             size 49.7
             lines 24.9
             cell width 10
             cell height 10
             label-offset 20
             color 257
             modeless
             title "Customers"
             handle window-handle.
          display screen-1.
          perform until exit-pushed
             accept scr-cust-code on exception
                perform is-screen-1-evaluate-func
             end-accept
          end-perform.
          destroy window-handle.
      ...
```

Format 7 - Accept to wait for an event to terminate it and then display the event

```
*> ws-event can be a pic 9(9).
accept event on exception ws-event continue
display message "Event : " ws-event
```

Format 8 - Accept from environment-value previously setup with display upon

```
display "envfullname" upon environment-name
display "Adam Smith" upon environment-value
accept f-name from environment-value
```

ADD

Format 1

```
ADD { {Identifier-1} } ... TO { Identifier-2 [ROUNDED] } ...
      {Literal-1   }
      {Class-1     }

  [ ON SIZE ERROR Imperative-Statement-1 ]

  [ NOT ON SIZE ERROR Imperative-Statement-2 ]

  [END-ADD]
```

Format 2

```
ADD { {Identifier-1} } ... TO { {Identifier-2} } ...
      {Literal-1   }           {Literal-2   }
      {Class-1     }           {Class-2     }

  GIVING { Identifier-3 [ROUNDED] } ...

  [ ON SIZE ERROR Imperative-Statement-1 ]

  [ NOT ON SIZE ERROR Imperative-Statement-2 ]

  [END-ADD]
```

Format 3

```
ADD { CORRESPONDING }{ {Identifier-4} } ... TO { {Identifier-5} } ...
    { CORR          }
  [ ON SIZE ERROR Imperative-Statement-1 ]

  [ NOT ON SIZE ERROR Imperative-Statement-2 ]

  [END-ADD]
```

Syntax rules

1. Identifier-1 and identifier-2 shall reference a numeric data item.
2. Literal-1 and literal-2 shall be numeric literals.
3. Class-1 and Class-2 shall be OBJECT-REFERENCE of Java numeric primitive types (i.e. "int") or objects (i.e. "java.lang.Integer")
4. Identifier-3 shall reference a numeric data item or a numeric-edited data item.
5. CORRESPONDING and CORR are synonymous.
6. Identifier-4 and identifier-5 shall reference a group item.

General rules

1. When Format 1 is used, the initial evaluation consists 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 operand is specified, the sum of such operands. The sum of the initial evaluation and the value of the data item referenced by identifier-2 is 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 is 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 is 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 consists 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 is 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 is 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. Additional rules and explanations relative to this statement are given in Overlapping operands; Incompatible data; ROUNDED phrase; SIZE ERROR phrase and size error condition; CORRESPONDING phrase; and Arithmetic statements.
4. In Format 3, each pair of corresponding elementary numeric items in Identifier-4 and Identifier-5 are added together. The results are moved to the corresponding items in Identifier-5.

Examples

Format 1- Increment the numeric data item num-var by 1

```
add 1 to num-var
```

Format 1 - Add 1.253 to num-var, producing a rounded result and check if the resulting value does not exceed the defined size of num-var.

```
add 1.253 to num-var rounded
 on size error
    display message "Value to long for num-var"
 not on size error
    display message "New value for num-var : " num-var
end-add
```

Format 2 - Assign to num-3 the result of adding num-1 and num-2.

```
add num-1 to num-2 giving num-3
```

Format 2 - Assign to num-3 the result of adding num-1 and num-2, checking if the resulting value does not exceed the defined size of num-3.

```
add num-1 to num-2 giving num-3
 on size error
    display message "Value to long for num-3"
 not on size error
    display message "New value for num-3 : " num-3
end-add
```

Format 3 - Add values from one group data item to another one with corresponding sub-items.

```
working-storage section.
01 work-hours.
   05 test-hours    pic 9(3) value 10.
   05 doc-hours     pic 9(3) value 11.
   05 support-hours pic 9(3) value 12.
01 additional-work-hours.
   05 test-hours    pic 9(3) value 2.
   05 doc-hours     pic 9(3) value 4.
   05 support-hours pic 9(3) value 2.

procedure division.
add-hours.
   add corresponding additional-work-hours to work-hours
       on size error display message "Error adding to work-hours"
       not on size error
           display message
                   test-hours    of work-hours " , "
                   doc-hours     of work-hours " , "
                   support-hours of work-hours
   end-add
```

ALTER

General Format

```
ALTER { Procedure-Name-1 TO [ PROCEED TO ] Procedure-Name-2 } ...
```

General Rules

1. Procedure-Name-1 must contain only a GO TO statement without DEPENDING clause.
2. The statement causes the GO TO statement in Procedure-Name-1 to transfer the control to Procedure-Name-2 instead of the declared paragraph.

Note - This statement is obsolete and not fully supported. In COBOL Programs the runtime will generate a warning, and in COBOL Classes, the runtime will generate a severe compiler error.

Examples

Compute simple or complex tax doing an alter of a go to.

```
main.
   perform tax-main thru tax-exit
   display message ws-tax
   goback.

 tax-main.
   if tax-type = "complex"
      alter go-to-compute-tax to proceed to compute-complex-tax
   end-if.

 go-to-compute-tax.
   go to compute-simple-tax.

 compute-complex-tax.
   compute ws-tax = ws-amount * ws-base-percent * ws-tax-rate
   go to tax-exit.

 compute-simple-tax.
   compute ws-tax = ws-amount * ws-tax-rate.

 tax-exit.
   exit.
```

ASSERT

General Format

```
ASSERT condition OTHERWISE value-1 [ value-2 ... value-n ]
```

Syntax Rules

1. *value-1*, *value-2* and *value-n* can be either literals or data items.

General Rules

1. The ASSERT statement is evaluated only when the program runs with the *-ea* Java option. In order to take advantage of assertions, you should run the COBOL program with one of the following commands

   ```
   i. java -ea PROG

   i. iscrun -J-ea PROG
   ```

2. If *condition* is true, the program continues to the next statement.
3. If *condition* is false, then a java.lang.AssertionError is raised. The error message is set to the combination of the values specified in the OTHERWISE clause.

Examples

Sample to verify age before continue program flow.

```
assert ws-age > 17 otherwise "Invalid Age"
display message "Welcome, your age has been verified!"
```

CALL

Format 1

```
CALL {CLIENT} Program-Name
     {IN THREAD}
  [ HANDLE IN Handle-1 ]

  [ USING { [ BY {REFERENCE} ] { {Parameter} } ... } ... ]
                {CONTENT   }     {OMITTED   }
                {VALUE     }

  [ {RETURNING} INTO Return-Val ]
    {GIVING   }

  [ ON {EXCEPTION} Statement-1]
       {OVERFLOW  }

  [ NOT ON {EXCEPTION} Statement-2]
           {OVERFLOW  }

  [END-CALL]
```

Format 2

```
CALL RUN Program-Name

  [ USING { [ BY {REFERENCE} ] { {Parameter} } ... } ... ]
                {CONTENT   }     {OMITTED   }
                {VALUE     }

  [ {RETURNING} INTO Return-Val ]
    {GIVING   }

  [ ON {EXCEPTION} Statement-1]
       {OVERFLOW  }

  [ NOT ON {EXCEPTION} Statement-2]
           {OVERFLOW  }

  [END-CALL]
```

Format 3

```
CALL PROGRAM Program-Name

  [ USING { [ BY {REFERENCE} ] { {Parameter} } ... } ... ]
                 {CONTENT  }     {OMITTED  }
                 {VALUE    }

  [ {RETURNING} INTO Return-Val ]
    {GIVING   }

  [ ON {EXCEPTION} Statement-1]
       {OVERFLOW }

  [ NOT ON {EXCEPTION} Statement-2]
           {OVERFLOW }

  [END-CALL]
```

Syntax Rules

All Formats

1. Program-Name shall be defined as an alphanumeric data item or String literal.
2. Parameter shall reference an address-identifier or a data item defined in the file, working-storage, and linkage. It can reference an arithmetic expression when passed BY VALUE.
3. Return-Val shall reference a data item defined in the file, working-storage or linkage.
4. The USING clause can appear either before or after the GIVING clause.

Formats 1 and 2

5. Return-Val shall not be object-reference.
6. If the BY REFERENCE phrase is specified or implied for a Parameter, it shall be neither a strongly-typed group item.

General Rules

All Formats

1. The instance of the program, or method that executes the CALL statement is the activating runtime element.
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. This 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.
3. Program-Name and Parameter are evaluated and item identification is done for Return-Val at the beginning of the execution of the CALL statement. If an exception condition exists, no program is called and execution proceeds as specified in general rule 6. If an exception condition does not exist, the value of identifier-2 is made available to the called program at the time control is transferred to that program.
4. The program being called is identified by its program-name or its location, which is determined as follows:
   A. If Program-Name references an alphanumeric or national data item, program-name is the program-name of the program being called

B. If the program being called is a COBOL program, the runtime system attempts to locate the program being called.

5. If a fatal exception condition has not been raised, the program specified by the CALL statement is made available for execution and control is transferred to the called program.
6. If the program was not successfully called the following actions occur:
   A. If the ON OVERFLOW or ON EXCEPTION phrase is specified in the CALL statement, control is transferred to Statement-1. Execution then continues according to the rules for each statement specified in 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 Statement-1, control is transferred to the end of the CALL statement and the NOT ON EXCEPTION phrase, if specified, is ignored.

      ii. If an ON OVERFLOW nor an ON EXCEPTION phrase is specified, an error is returned

   B. If the program was successfully called, after control is returned from the called program the ON OVERFLOW or ON EXCEPTION phrase, if specified, is ignored. Control is transferred to the end of the CALL statement or, if the NOT ON EXCEPTION phrase is specified, to Statement-2. If control is transferred to Statement-2, execution continues according to the rules for each statement specified in Statement-2.
7. If a RETURNING phrase is specified, the result of the activated program is placed into Return-Val.
8. BY CONTENT, BY REFERENCE and BY VALUE phrases are transitive across the parameters that follow them until another BY CONTENT, BY VALUE or BY REFERENCE phrase is encountered. If neither the BY CONTENT nor the BY VALUE nor the BY REFERENCE phrase is specified prior to the first parameter, the BY REFERENCE phrase is assumed.
9. BY REFERENCE causes the address of the data item to be passed to the receiving program. BY CONTENT and BY VALUE cause the address of a copy of the data item to be sent, therefore any changes made to the parameter in the called program are not seen by the caller. When passing parameters to C functions with either BY CONTENT or BY REFERENCE clauses, isCOBOL passes the address of the data item, unless the data item is an integer, in that case the item value is passed.
10. When a literal is passed, it's passed as a USAGE DISPLAY data item whose size is the size of the literal, unless the program is compiled with the -ccmf flag, in that case the literal is treated as follows:
    - BY REFERENCE: A numeric literal is passed as an integer 32-bit comp-x item if it is not negative or as an integer 32-bit comp (binary) item if it is negative, even if the literal is not an integer. A decimal point in the numeric literal is ignored. An alphanumeric literal is passed in the same way as when the BY CONTENT clause is used.
    - BY VALUE: If a numeric literal is specified, then the data description of the item is equivalent to a signed numeric item USAGE BINARY.
    - BY CONTENT: if a literal is specified, then the implied data description of the item is equivalent to an alphanumeric data item with the same size as the literal and with its contents set to the value of the literal.
11. When a resource property is passed, it's passed as a USAGE NATIONAL data item whose size is the size of the property name (e.g. twice as the number of characters in the property name).
12. Dynamic length items are always passed BY REFERENCE.
13. The program identified by *Program-Name* is searched among the paths specified by the iscobol.code_prefix setting. If the code-prefix is not set, then the program is searched in the Classpath.
14. Extensions in *Program-Name* are automatically stripped by the runtime.
15. Paths in *Program-Name* are considered only if iscobol.code_prefix is set. Relative paths in *Program-Name* are appended to the code-prefix paths.
16. You can install a hook class for the CALL statement via the iscobol.call_cancel.hook configuration property.
17. A special register named RETURN-CODE is automatically created by the compiler and is shared by all

programs of a run unit. This special register is defined as:

```
77  RETURN-CODE  SIGNED-LONG EXTERNAL.
```

If you call a C program , the return value of the C function is placed into this register. If you call the SYSTEM library routine, the status of the call is placed into this register. The verbs EXIT, STOP, and GOBACK can also place a value into the RETURN-CODE register. The compiler also creates an unsigned version of the return code called RETURN-UNSIGNED. It has the following implied definition:

```
77  RETURN-UNSIGNED REDEFINES RETURN-CODE UNSIGNED-LONG EXTERNAL.
```

If the RETURNING phrase is used, then the return value of the called program is moved to *Return-Val*. This is accomplished by the following rule:

- If *Return-Val* is a signed data item, then the value of RETURN-CODE is moved to *Return-Val*.
- If *Return-Val* is unsigned, then RETURN-UNSIGNED is moved instead.

You should avoid using RETURN-CODE or RETURN-UNSIGNED for *Return-Val*. This is pointless because, after assigning a value to *Return-Val*, the CALL statement restores the previous value of RETURN-CODE.

Format 1

1. The runtime framework searches for a called subroutine in this order:
   A. If the name is the exact name of a shared library then that shared library is loaded into memory. Otherwise
   B. the entry points in the already loaded user subroutines are searched. If not found
   C. the statically declared C functions are searched. If not found
   D. the C functions available in the already loaded shared libraries are searched. If not found
   E. the isCOBOL library subroutines are searched. If not found
   F. the user subroutines are searched. If not found
   G. the runtime decorates the subroutine name with a shared library name (e.g. *foo* may become *foo.dll* or *libfoo.so*, depending on the O.S.) and tries to load it. If the library is successfully loaded, then the runtime looks for a function with the same name (*foo*), otherwise
   H. the remote subroutines available, if any, are searched.

   Note - The above search is only performed the first time the subroutine is called. Once found, the runtime keeps track of it's location. The next time the same subroutine is called, the runtime looks for it directly in the right place. This improves the performance of the CALL statement.
2. The THREAD clause makes the called program run asynchronously with the calling program.
   When a program is called asynchronously, it's good practice to pass parameters BY VALUE. Passing parameters BY REFERENCE may produce unexpected results.
   Return-Val is set with the called program exit status when the thread terminates.
   When a program is called asynchronously, the runtime looks for it only locally ignoring iscobol.remote.code_prefix.
3. The CLIENT clause is used when running in ApplicationServer mode to execute the called program on a client machine. The called program on the client can be either a COBOL program, a library routine or a C function. If running without ApplicationServer, the CLIENT clause is ignored.
   A program compiled with -cp option can call a C function with a CALL CLIENT statement as long as the arguments are not numeric data-items passed BY VALUE and are not pointers. If the C function requires either numeric data-items passed BY VALUE or pointers, it's possible to use the CALL CLIENT statement to call a COBOL program compiled with -cp option on the client PC and have this COBOL program call the C function.
4. On AIX systems some libraries (such as libdb2.a) can be called by dynamically specifying the member names.

To achieve it with isCOBOL, use the following syntax:

```
CALL "/lib/foo.a(member.o)"
```

5. On Windows systems it is possible to specify the calling convention when the DLL library is loaded. The convention can be specified by adding a slash character followed by the desired convention at the end of the DLL name. Use the following syntax to load a DLL library with CDECL convention:

```
CALL "foo.dll/0"
```

Use the following syntax to load a DLL library with PASCAL convention:

```
CALL "foo.dll/1"
```

The choosen convention will be adopted by the runtime when calling the functions stored in the loaded DLL library, regardless of the iscobol.dll_convention setting.

Format 2

1. The CALL RUN statement makes the called program run synchronously or asynchronously in a separate thread. Return-Val is set with the called program exit status when the thread terminates.
2. The called program doesn't inherit the environment set by the calling program, but starts in a clean environment instead.
3. The synchronicity is controlled by the iscobol.call_run.sync (boolean) configuration setting. By default, this call is asynchronous.

Format 3

1. CALL PROGRAM works as the CHAIN statement. It causes the current run unit to terminate and initiates a new run unit, but in this case USING parameters are passed to data items specified in the Linkage Section.

Recursive calls

A program may directly or indirectly call itself. Such a CALL statement is considered a recursive call. By default, isCOBOL shares the program data with all recursive calls. Set iscobol.recursion_data_global (boolean) * to false in order to make isCOBOL create a new copy of data for each instance of the program.

Examples

Format 1 - Call a program passing parameters, receiving return code and validating exception.

```
call "program1" using parm-1 parm-2
  giving ret-code
  on exception display message "Could not call program1"
end-call
```

Format 1- Call a program on a different thread to run it in parallel with calling program saving the thread handle in p1-handle.

```
working-storage section.
77 p1-handle  usage handle of thread.
...
procedure division.
main.
  call thread "program1" handle in p1-handle
```

Format 1 - Call a program client side when running from thin-client

```
call client "program1"
```

Format 2 - Call a program and run it in parallel with current one (on different thread).

```
call run "program1"
```

Format 3 - Call a program terminating the current run unit and starting a new one

```
call program "program1"
```

CANCEL

Format 1

```
CANCEL [CLIENT] { {Program-Name} } ...
```

Format 2

```
CANCEL [CLIENT] ALL
```

Syntax rules

1. Program-Name shall be defined as an alphanumeric or national data item.

General rules

1. The program-name is used by the runtime system to locate the program.
2. The program to be canceled shall not be in the active state or contain a program in the active state. If a program in the active state is explicitly or implicitly referenced in a CANCEL statement, the referenced program is not canceled.
3. 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.
4. The contents of data items in external data records described by a program are not changed when that program is canceled.
5. The CANCEL statement has no effect on a program with the RESIDENT clause specified in its PROGRAM-ID paragraph.
6. The ALL clause affects all programs in the non-active state as described in rule 2. It also causes the

disconnection from the isCOBOL Application Server if a CALL to a remote object was performed before.

7. The CLIENT clause affects the client called program (with CALL CLIENT statement) when running in ApplicationServer mode.
8. The CANCEL statement doesn't have any effect on programs called through CALL THREAD. Threads are automatically cancelled by the Framework when they terminate.
9. The CANCEL statement doesn't have any effect on native libraries (e.g. DLLs). Call the C$UNLOAD_NATIVE library routine in order to cancel a native library.
10. You can install a hook class for the CANCEL statement via the iscobol.call_cancel.hook configuration property.

Examples

Format 1- Cancel a program that is not in active state.

```
call "program1"
cancel "program1"
```

Format 1 - Cancel a program executed client side when running on thin-client.

```
call client "program1"
cancel client "program1"
```

Format 2 - Cancel all programs that are not in active state.

```
call "program1"
call "program2"
call "program3"
cancel all
```

Format 2 - Cancel all client side executed programs, when running on thin-client that are not in active state.

```
call client "program1"
call client "program2"
call client "program3"
cancel client all
```

CHAIN

General Format

```
CHAIN program-name [ USING {parameter} ... ]

  [ ON {EXCEPTION} statement ]
       {OVERFLOW }

  [ END-CHAIN ]
```

Syntax Rules

1. *program-name* is a nonnumeric literal or an alphanumeric data item.
2. *Parameter* is any data item including level-88 or a nonnumeric literal. There are no limits on the number of *parameters* that may be specified.
3. *Statement* is an imperative statement to be used if the CHAINed program can not be loaded.

General Rules

1. program-name is executed in a new logical run unit.
2. If the USING phrase is specified, each parameter is transferred to *program-name* and is mapped to the corresponding CHAINING argument in the PROCEDURE DIVISION phrase.

Exmaples

Call a program terminating the current run unit and starting a new one

```
chain "program1"
```

CLOSE

Format 1

```
CLOSE { File-Name [REEL] [ WITH {NO REWIND} ] } ...
                                {LOCK     }
```

Format 2

```
CLOSE WINDOW Window-Handle [ WITH NO DISPLAY ]
```

Format 3

```
CLOSE POP-UP wcb [CONTROL Cntrl-String]
```

Syntax rules

General rules

1. The `REEL` clause is treated as a commentary.
2. The file connector referenced by File-Name shall be open. If the file connector is not open, the CLOSE statement is unsuccessful and the I-O status indicator for the file connector is set based on your FileStatus setting (iscobol.file.status *).
3. The NO REWIND clause causes the current unit to be left in its current position.
4. When the LOCK clause is specified, the file connector referenced by file-name shall not be opened again during the execution of this run unit.
5. The execution of the CLOSE statement causes the value of the I-O status associated with file-name to be updated.
6. The file lock and any record locks associated with the file connector referenced by file-name are released by the execution of the CLOSE statement.
7. If more than one file-name 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 in the same order as specified in the CLOSE statement.
8. The Format 2 CLOSE has no effect on the main window, that is destroyed only when the runtime terminates.
9. The Format 3 CLOSE closes a window created by a Format 14 DISPLAY statement. The CONTROL is ignored.

Examples

Format 1 - Close file

```
close file1
```

Format 1- Close file and leave it locked for the rest of the run unit.

```
close file1 lock
```

Format 2 - Display and accept screen on graphical window and then close the window

```
       working-storage section.
       01 window-handle usage handle.
       01 cust-values.
          05 ws-cust-code  pic x(5).
          05 ws-cust-name  pic x(50).

       screen section.
       01 screen-1.
          03 scr-cust-code Entry-Field
             using ws-cust-code
             line 3.0
             column 21.4
             size 12.7 cells
             lines 3.8 cells
             id 1
             3-d.
          03 scr-cust-name Entry-Field
             using ws-cust-name
             line 8.7
             column 21.4
             size 24.5 cells
             lines 4.6 cells
             id 2
             3-d.
          03 scr-lab-1 Label
             line 2.7
             column 4.5
             size 13.5 cells
             lines 3.7 cells
             id 3
             title "Code :".
          03 scr-lab-2 Label
             line 9.3
             column 4.5
             size 13.5 cells
             lines 3.7 cells
             id 4
             title "Name :".
          03 scr-pb-save Push-Button
             line 16.4
             column 15.7
             size 15.5 cells
             lines 6.4 cells
             id 5
             title "Save".
         ...
```

```
      procedure division.
      display-and-accept.
         display standard  window background-low
             screen line 41
             screen column 91
             size 49.7
             lines 24.9
             cell width 10
             cell height 10
             label-offset 20
             color 257
             modeless
             title "Customers"
             handle window-handle.
          display screen-1.
          perform until exit-pushed
             accept screen-1 on exception
                perform is-screen-1-evaluate-func
             end-accept
          end-perform.
          close window window-handle.
      ...
```

COMMIT

General Format

```
COMMIT TRANSACTION
```

General rules

1. When this statement is executed, all locked records owned by the current transaction are unlocked.
2. The function of COMMIT depends on whether or not it ends a transaction. For file systems used via EXTFH property handler linked with the runtime, each system's native mechanism for transaction management is invoked.
3. When ROLLBACK is enabled in the FILE-CONTROL entry for a file, the record and file locking rules are extended for that file. Updated records in a transaction are locked until COMMITed or cancelled with the ROLLBACK statement.
4. During a transaction involving indexed or relative files, a CLOSE of a file that is locked, or that has locked records, is postponed until the transaction is committed or rolled back.

Examples

Commit a transaction after several records were modified

```
*> This sample will work only with a file system that supports transactions
*> The select should include lock clause with rollback: lock automatic with rollback
...
input-output section.
file-control.
select invoices assign to inv-path
   organization indexed
   access dynamic
   record key cust-code
   lock mode manual with rollback |the rollback clause is required to
   status inv-status.             |activate transaction management
...
procedure division.
...
   start transaction
   display "Customer to apply 10% discount? "
   accept ws-cust-code
   move ws-cust-code to cust-code
   read customers
        invalid key display message "Customer not found"
                    rollback
                    exit paragraph
   end-read
   accept ws-date from date
   move ws-date to cust-last-date-discount
   rewrite cust-rec
   move ws-cust-code to inv-cust-code
   start invoices key = inv-cust-code
         invalid key set end-of-invoices to true
         not invalid key set end-of-invoices to false
   end-start
   perform until end-of-invoices
     read invoices next at end exit perform
          not at end
              if inv-cust-code not = ws-cust-code
                 exit perform
              end-if
     end-read
     if inv-paid-status = "N"
        move 10 to inv-discount
        rewrite invoice-rec
     end-if
   end-perform
   display "Changes applied, confirm Commit: (Y/N)"
   accept apply-commit
   if apply-commit = "Y"
      commit *> All changes are definitely applied
   else
      rollback *> All changes get undone
   end-if.
```

COMPUTE

General Format

```
COMPUTE { Result-1 [ROUNDED]  } ... = Arithmetic-Expression

  [ ON SIZE ERROR Imperative-Statement-1 ]

  [ NOT ON SIZE ERROR Imperative-Statement-2 ]

  [END-COMPUTE]
```

Syntax rules

1. Result-1 shall reference either an elementary numeric item or an elementary numeric-edited item.

General rules

1. An arithmetic expression consisting of a single fixed-point numeric literal or a single fixed-point numeric data item evaluates to the exact value of that literal or identifier, before the application of any rounding, truncation, or decimal point alignment applicable for the COMPUTE statement and mode of arithmetic in effect.
2. Evaluation consists of determining the value of the arithmetic expression.

Examples

Compute arithmetic expression, leave the result in num-1

```
compute num-1 = num-2 * interest-rate / 100 / 12
```

Compute arithmetic expression, leave rounded result in num-1 and validate if result fits in num-1 defined size.

```
compute num-1 rounded = num-2 * interest-rate / 100 / year-days
  on size error display message "The last computation did not fit in num-1!"
  not on size error display message "new value of num-1 : " num-1
end-compute
```

CONTINUE

General Format

```
CONTINUE
```

Syntax rules

1. The CONTINUE statement may be used anywhere a conditional statement or an imperative-statement may be used.

General rules

1. The CONTINUE statement has no effect on the execution of the runtime element.

Examples

Continue in the next statement if condition is true

```
if ws-age > 17
   continue
else
   display message "Not valid age to use this Credit Card Verification App!"
end-if
display message "Welcome to our Credit Card Verification App!"
```

DELETE

Format 1

```
DELETE File-Name RECORD

  [ INVALID KEY Statement-1 ]

  [ NOT INVALID KEY Statement-2 ]

  [END-DELETE]
```

Format 2

```
DELETE FILE File-Name [ File-Name-2 ... ]
```

Format 3

```
DELETE File-Name RECORD
  [ KEY IS DataName-1 ]
```

Syntax rules

1. The Format 1 DELETE statement cannot be specified for a file with sequential or XML organization.
2. The INVALID KEY and the NOT INVALID KEY phrases shall not be specified for a DELETE statement that references a file that is in sequential access mode.
3. The Format 3 DELETE statement can be used only for a file with XML organization.
4. *DataName-1* can be qualified, and references a data item whose declaration includes an IDENTIFIED BY clause and is included in the XD record declarations for *File-Name*.

General rules

Format 1

1. The open mode of the file connector referenced by file-name-1 shall be I-O and the physical file associated with that file connector shall be a mass storage file.
2. For a file that is 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 physical file the record that was accessed by that

READ statement.

3. If the file is indexed and the access mode is random or dynamic, the mass storage control system (MSCS) logically removes from the physical file the record identified by the content of the prime record key data item associated with file-name-1. If the physical file does not contain the record specified by the key, the invalid key condition exists.
4. If the file is relative and the access mode is random or dynamic, the mass storage control system (MSCS) logically removes from the physical file that record identified by the content of the relative key data item associated with file-name-1. If the physical file does not contain the record specified by the key, the invalid key condition exists.
5. After the successful execution of a DELETE statement, the identified record has been logically removed from the physical file and can no longer be accessed.
6. The execution of a DELETE statement does not affect the content of the record area or the content of the data item referenced by the data-name specified in the DEPENDING ON phrase of the RECORD clause associated with file-name-1.
7. The file position indicator is not affected by the execution of a DELETE statement.
8. The execution of the DELETE statement causes the value of the I-O status associated with file-name-1 to be updated.
9. Transfer of control following the successful or unsuccessful execution of the DELETE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the DELETE statement.

Format 2

10. DELETE FILE removes File-Name from the system. File-Name must be closed before execute the DELETE FILE statement.

Format 3

11. A Format 3 DELETE statement deletes an XML element and all sub-elements from the in-memory representation of an XML document. No stream I/O is actually performed by the DELETE statement.
12. When KEY is specified, this deletes the internal representation of the elements and child elements associated with *DataName-1*. All other elements represented in the record remain unchanged.
13. If the internal representation of the XML document is modified through the use of WRITE KEY, REWRITE KEY, or DELETE KEY, and the internal representation is cleared with either a CLOSE or READ (no key) statement, then the CLOSE or READ statement will return a status of -10, indicating the operation succeeded but no write was done.

Examples

Format 1 - Delete record by key and validate invalid key

```
  move 5432 to cust-code
  delete cust-file
    invalid key display message "Key not found to delete! : " cust-code
    not invalid key display message "Successfully deleted customer : " cust-code
  end-delete
```

Format 2 - Delete temporary file after done with it.

```
open output cust-tmp-file
...
close cust-tmp-file
delete file cust-tmp-file
```

DESTROY

Format 1

```
DESTROY { Screen-Name-1 } ... [ UPON Screen-Group ]
        { Handle-1      }
```

Format 2

```
DESTROY ALL CONTROLS
```

Format 3

```
DESTROY CONTROL
```

Remaining phrases are optional, can appear in any order.

```
  AT Screen-Location

  AT LINE NUMBER Line-Num [CELL  ]
                          [CELLS ]
                          [PIXEL ]
                          [PIXELS]

  AT {COLUMN  } NUMBER Col-Num [CELL  ]
     {COL     }                [CELLS ]
     {POSITION}                [PIXEL ]
     {POS     }                [PIXELS]
```

Syntax rules

1. *Screen-name-1* is the name of a screen description entry found in the Screen Section.
2. *Handle-1* is a USAGE HANDLE or PIC X(10) data item.
3. *Screen-location* is an integer data item or literal that contains exactly 4 or 6 digits.
4. *Line-num* and *col-num* are numeric data items or literals. These may contain non-integer values.
5. *Screen-group* is a group item in Screen Section, so a Screen-name that is not associated to a control-class.

General rules

Format 1

1. The DESTROY verb clears the screen of active controls and removes the assigned handle(s) from memory and sets the value to binary zeroes.
2. If *Handle-1* refers to a sub-window, the result is the same as a CLOSE WINDOW, causing all controls on that window to be DESTROYed. All child windows are also DESTROYed.
3. If *Handle-1* is a handle of a menu or layout manager, the controlling window should be DESTROYed before the handle
4. The DESTROY verb as no effect on the main window, that is destroyed only when the runtime terminates.
5. If *Handle-1* is a handle of bitmap or menu, the DESTROY verb has no effect. Use WBITMAP-DESTROY to destroy a bitmap handle and WMENU-DESTROY to destroy a menu handle.
6. If *Handle-1* is a handle of one of the system pre-defined font (fixed-font, traditional-font, default-font, small-

font, medium-font or large-font) the DESTROY verb has no effect.

7. If the UPON clause is followed by a Screen-group, the item is removed from that group and the screen is updated to reflect the change.

Format 2

1. DESTROY ALL CONTROLS destroys all controls created for the current window.

Format 3

1. Format 3 DESTROY uses the LINE/COLUMN phrases to indicate which control to DESTROY.

Examples

Format 1 - Destroy a screen.

```
display input-screen-1
perform until esc-pressed
  accept input-screen-1
end-perform
destroy input-screen-1
```

Format 1 - Destroy window.

```
display initial  window
        screen line 10 screen column 10
        size 40.0 lines 25.0
        cell width 10 cell height 10
        title "Customer Information"
        handle cust-win-handle
...
destroy cust-win-handle
```

Format 3 - Destroy controls by line and col position.

```
display entry-field line 5.0 col 20.0
display label line 5.0 col 3.0 title "Customer code: "
...
destroy control line 5.0 col 3.0
destroy control line 5.0 col 20.0
```

DISPLAY

Format 1

```
DISPLAY {identifier-1} ... [ UPON mnemonic-name-1 ] [WITH NO ADVANCING] [END-DISPLAY]
        {literal-1   }
```

Format 2

```
DISPLAY screen-name-1

  [ AT { LINE NUMBER { identifier-2 } ]
                     { integer-1    }

  [ AT { COLUMN } NUMBER { identifier-3 } ]
       { COL    }        { integer-2    }

  [UPON { Tab-Control-Handle (Page-Index) } ]
        { Ribbon-Handle (Page-Index)      }
        { New-Window                      }
        { Screen-Group                    }

[END-DISPLAY]
```

Format 3

```
DISPLAY [ ( Line-Num, Col-Num ) ] { Src-Item [ UPON New-Window ] }  [ END-DISPLAY ]
                                  { OMITTED                     }
```

Remaining phrases are optional, can appear in any order

```
  AT Screen-Loc

  AT LINE NUMBER Line-Num

  AT {COLUMN  } NUMBER Col-Num
     {COL     }
     {POSITION}
     {POS     }
```

```
WITH SIZE Length

WITH NO ADVANCING

{ERASE} [TO END OF] {LINE  }
{BLANK}             {SCREEN}

{ERASE} [EOS]
{BLANK} [EOL]

WITH  {BELL}
      {BEEP}

{UNDERLINED}
{HIGHLIGHT }
{HIGH      }
{BOLD      }
{LOWLIGHT  }
{LOW       }
{STANDARD  }

WITH {BLINKING}
     {BLINK   }

{REVERSE-VIDEO}
{REVERSE      }
{REVERSED     }

SAME

WITH {COLOR } Color-Val
     {COLOUR}

{FOREGROUND-COLOR } IS Fg-Color
{FOREGROUND-COLOUR}

{BACKGROUND-COLOR } IS Bg-Color
{BACKGROUND-COLOUR}

SCROLL [UP  ] [ BY Scrl-Num {LINE } ]
       [DOWN]               {LINES}

OUTPUT {JUSTIFIED} {LEFT    }
       {JUST     } {RIGHT   }
                   {CENTERED}

WITH {CONVERSION}
     {CONVERT   }

CONTROL Cntrl-String
```

Format 4

```
DISPLAY {SUBWINDOW} [ UPON New-Window ] [END-DISPLAY]
        {WINDOW   }
```

Remaining phrases are optional, can appear in any order.

```
  AT screen-loc

  AT LINE NUMBER Line-Num

  AT {COLUMN  } NUMBER Col-Num
     {COL     }
     {POSITION}
     {POS     }

  SIZE Length

  LINES Height

  {ERASE} SCREEN
  {BLANK}

  {REVERSE-VIDEO}
  {REVERSE      }
  {REVERSED     }

  WITH {COLOR } Color-Val
       {COLOUR}

  {FOREGROUND-COLOR } IS Fg-Color
  {FOREGROUND-COLOUR}

  {BACKGROUND-COLOR } IS Bg-Color
  {BACKGROUND-COLOUR}

  {HIGHLIGHT}
  {HIGH     }
  {BOLD     }
  {LOWLIGHT }
  {LOW      }
  {STANDARD }

  {BACKGROUND-HIGH    }
  {BACKGROUND-LOW     }
```

```
  {BACKGROUND-STANDARD}

  BOXED

  SHADOW

  [TOP   ] [CENTERED] TITLE IS Title
  [BOTTOM] [LEFT    ]
           [RIGHT   ]

  WITH NO SCROLL

  WITH NO WRAP

  CONTROL VALUE IS Control-Val

  POP-UP AREA IS Save-Area
```

Format 5

```
DISPLAY SCREEN SIZE {80 }
                    {132}

  [ ON EXCEPTION statement-1 ]

  [ NOT ON EXCEPTION statement-2 ]

[ END-DISPLAY ]
```

Format 6

```
DISPLAY LINE [ UPON New-Window ] { SIZE Length  } [END-DISPLAY]
                                 { LINES Height }
```

Remaining phrases are optional, can appear in any order.

```
  AT screen-loc

  AT LINE NUMBER Line-Num

  AT {COLUMN  } NUMBER Col-Num
     {COL     }
     {POSITION}
     {POS     }

  {REVERSE-VIDEO}
  {REVERSE      }
  {REVERSED     }

  WITH {COLOR } Color-Val
       {COLOUR}

  [CENTERED] TITLE IS title
  [LEFT    ]
  [RIGHT   ]
```

Format 7

```
DISPLAY BOX [ UPON New-Window ][END-DISPLAY]
```

Remaining phrases are optional, can appear in any order.

```
  AT screen-loc

  AT LINE NUMBER Line-Num


  AT {COLUMN  } NUMBER Col-Num
     {COL     }
     {POSITION}
     {POS     }

  SIZE Length

  LINES Height

  {REVERSE-VIDEO}
  {REVERSE      }
  {REVERSED     }

  WITH {COLOR } Color-Val
       {COLOUR}

  [TOP   ] [CENTERED] TITLE IS title
  [BOTTOM] [LEFT    ]
           [RIGHT   ]
```

Format 8

```
DISPLAY Source UPON WINDOW [ { TOP    } ] [ { CENTERED }] TITLE  [END-DISPLAY]
                             { BOTTOM }     { LEFT     }
                                            { RIGHT    }
```

Format 9

```
DISPLAY Title-1 UPON { FLOATING WINDOW Handle-1 } TITLE [END-DISPLAY]
                     { GLOBAL WINDOW            }
```

Format 10

```
DISPLAY [ {DOCKABLE   } [GRAPHICAL] ] WINDOW
          {DOCKING    }
          {FLOATING   }
          {INDEPENDENT}
          {INITIAL    }
          {MDI-CHILD  }
          {MDI-PARENT }
          {STANDARD   }
  [ HANDLE IN Window-Handle ]
```

The following are optional phrases:

```
{MODELESS}
{MODAL   }

{LINK} TO THREAD
{BIND}

SCREEN LINE NUMBER screen-line

SCREEN {COLUMN  } NUMBER screen-col
       {COL     }
       {POSITION}
       {POS     }

SCREEN-INDEX screen-idx

AT screen-loc

AT LINE NUMBER line-num

AT {COLUMN  } NUMBER col-num
   {COL     }
   {POSITION}
   {POS     }
```

```
SIZE   length

LINES  height

FONT {IS} font-1
     {= }

CONTROL FONT {IS} font-3
             {= }

CELL
 {SIZE  }  [IS] {cell-units                                    }
 {HEIGHT}  [= ] {control-type-name  FONT font-2 [SEPARATE  ]}
 {WIDTH }       {control-type-name  FONT        [OVERLAPPED]}

{ERASE} SCREEN
{BLANK}

{REVERSE-VIDEO}
{REVERSE      }
{REVERSED     }

WITH {COLOR } color-val
     {COLOUR}

{FOREGROUND-COLOR } IS fg-color
{FOREGROUND-COLOUR}

{BACKGROUND-COLOR } IS bg-color
{BACKGROUND-COLOUR}

GRADIENT-COLOR-1  IS gd-color-1
GRADIENT-COLOR-2  IS gd-color-2
GRADIENT-ORIENTATION  IS gd-orientation

{HIGHLIGHT}
{HIGH     }
{BOLD     }
{LOWLIGHT }
{LOW      }
{STANDARD }

{BACKGROUND-HIGH    }
{BACKGROUND-LOW     }
{BACKGROUND-STANDARD}

{ [USER-GRAY] [USER-WHITE] }
{        USER-COLORS       }

BOXED

SHADOW

TITLE-BAR

[TOP   ] [CENTERED] TITLE IS title
[BOTTOM] [LEFT    ]
         [RIGHT   ]

WITH SYSTEM MENU

WITH NO SCROLL
```

```
 WITH NO WRAP

 {NO-CLOSE}

 {AUTO-RESIZE}

 {RESIZABLE  }

 MIN-SIZE  {= } min-size
           {IS}
 MAX-SIZE  {= } max-size
           {IS}
 MIN-LINES {= } min-lines
           {IS}
 MAX-LINES {= } max-lines
           {IS}

 CONTROL VALUE {IS} control-val
               {= }

 LAYOUT-MANAGER {IS} manager
                {= }

 VISIBLE {IS} {TRUE         }
          {= } {FALSE        }
               {visible-state}

 POP-UP MENU {IS}  {menu-1}
             {= }  {NULL}

 {POP-UP AREA IS } handle-name
 {HANDLE {IS}    }
         {IN}

 CONTROLS-UNCROPPED

 EVENT PROCEDURE IS { proc-1 [ {THROUGH} proc-2 ] }
                            {THRU   }
                    { NULL                       }

 ACTION {IS} action
        {= }

 LAYOUT {IS} layout
        {= }
 UNDECORATED
```

Format 11

```
DISPLAY MESSAGE BOX { Text } ...
```

Remaining-Phrases are optional, can appear in any order.

```
  CENTERED

  TITLE {IS} Title
        {= }

  TYPE    {IS} Type
          {= }
  DEFAULT {IS} default
          {= }

  ICON {IS} icon
       {= }

  FONT {IS} font-1
       {= }

{ COLOR            } {IS} color-val
                     {= }
 {FOREGROUND-COLOR } {IS} fg-color
 {FOREGROUND-COLOUR} {= }
 {BACKGROUND-COLOR } {IS} bg-color
 {BACKGROUND-COLOUR} {= }

  BEFORE TIME  Timeout

  {GIVING   } Value
  {RETURNING}
```

Format 12

```
DISPLAY name UPON { ENVIRONMENT-NAME   } [END-DISPLAY]
                  { ENVIRONMENT-VALUE  }

  [ ON EXCEPTION Statement-1 ]

  [ NOT ON EXCEPTION Statement-2 ]

[END-DISPLAY]
```

Format 13

```
DISPLAY {control-class  }

[ title ]

  [ { UPON New-Window         } ]
    { UPON Grid-Handle (y, x) }
    { UPON Screen-Group       }

  [ HANDLE IN Control-Handle ]

The following are optional phrases:

  { { Property-Name          } [IS ] { [MULTIPLE] Property-Value [ LENGTH {IS} Length-
1 ] [ GIVING Result-1 ] } } ...
    { PROPERTY Property-
Type } [ARE] { [TABLE   ]                          {= }
  }
                                [=  ]

  { Style-Name } ...

  AT screen-loc

  AT LINE NUMBER line-num

  AT {COLUMN  } NUMBER col-num
     {COL     }
     {POSITION}
     {POS     }

  SIZE   length

  LINES  height

  FONT {IS} font-1
       {= }

  WITH {COLOR } color-val
       {COLOUR}

  {FOREGROUND-COLOR } IS fg-color
  {FOREGROUND-COLOUR}

  {BACKGROUND-COLOR } IS bg-color
  {BACKGROUND-COLOUR}

  {BACKGROUND-HIGH    }
  {BACKGROUND-LOW     }
  {BACKGROUND-STANDARD}

  VISIBLE {IS} {TRUE         }
          {= } {FALSE        }
               {visible-state}

  EVENT PROCEDURE IS { proc-1 [ {THROUGH} proc-2 ] }
                                {THRU   }
                     { NULL                        }
```

Format 14

```
DISPLAY POP-UP wcb [CONTROL Cntrl-String]
```

Format 15

```
DISPLAY NOTIFICATION WINDOW
   [ HANDLE IN Window-Handle ]
```

The following are optional phrases:

```
SCREEN-INDEX screen-idx

SIZE   length

LINES  height

FONT {IS} font-1
     {= }

CONTROL FONT {IS} font-3
             {= }

CELL
 {SIZE  }  [IS] {cell-units                                    }
 {HEIGHT}  [= ] {control-type-name  FONT font-2 [SEPARATE  ]}
 {WIDTH }       {control-type-name  FONT        [OVERLAPPED]}

{ERASE} SCREEN
{BLANK}

{REVERSE-VIDEO}
{REVERSE      }
{REVERSED     }

WITH {COLOR } color-val
     {COLOUR}

{FOREGROUND-COLOR } IS fg-color
{FOREGROUND-COLOUR}

{BACKGROUND-COLOR } IS bg-color
{BACKGROUND-COLOUR}

GRADIENT-COLOR-1  IS gd-color-1
GRADIENT-COLOR-2  IS gd-color-2
GRADIENT-ORIENTATION  IS gd-orientation

{HIGHLIGHT}
{HIGH     }
{BOLD     }
{LOWLIGHT }
{LOW      }
{STANDARD }

{BACKGROUND-HIGH    }
{BACKGROUND-LOW     }
{BACKGROUND-STANDARD}

{ [USER-GRAY] [USER-WHITE] }
{        USER-COLORS       }

TOP
RIGHT
BOTTOM
LEFT

 VISIBLE {IS} {TRUE         }
         {= } {FALSE        }
              {visible-state}

BEFORE TIME  Timeout
```

Syntax rules

1. COLUMN, COL, POSITION and POS are synonymous.
2. ERASE and BLANK are synonymous.
3. BELL and BEEP are synonymous.
4. HIGHLIGHT, HIGH and BOLD are synonymous.
5. LOWLIGHT and LOW are synonymous.
6. BLINKING and BLINK are synonymous.
7. REVERSE-VIDEO, REVERSE and REVERSED are synonymous.
8. COLOR and COLOUR are synonymous.
9. FOREGROUND-COLOR and FOREGROUND-COLOUR are synonymous.
10. BACKGROUND-COLOR and BACKGROUND-COLOUR are synonymous.
11. LINE and LINES, in the SCROLL phrase, are synonymous.
12. JUSTIFIED and JUST are synonymous.
13. CONVERSION and CONVERT are synonymous.
14. GIVING and RETURNING are synonymous.
15. *Mnemonic-name-1* can be SYSOUT, SYSERR, CONSOLE or CRT. CONSOLE is synonym of SYSOUT.

Format 1

16. Mnemonic-name-1 shall be associated with a hardware device in the SPECIAL-NAMES paragraph in the environment division.

Format 2

17. Identifier-2 and identifier-3 shall be unsigned integer data items.
18. Tab-Control-Handle is a USAGE HANDLE OF TAB-CONTROL data item or the screen name of a tab-control.
19. Page-Index is a numeric data item or literal.
20. Ribbon-Handle is a USAGE HANDLE OF RIBBON data item or the screen name of a ribbon.

Format 3

21. Src-Item is a literal or data item. It must be USAGE DISPLAY unless the CONVERSION phrase is also specified. Src-Item specifies the data to be displayed.
22. New-Window is a USAGE HANDLE OF WINDOW or PIC X(10) data item.
23. Screen-Loc is an integer data item or literal containing exactly 4 or 6 digits. It may also be a group item of 4 or 6 characters. If a numeric item is used, it must be a non-negative integer.
24. Line-Num, Col-Num, and Length are numeric data items or literals. They may be non-integer values. You can also specify the value with an arithmetic expression.
25. Color-Val and Scrl-Num are numeric data items or literals. Color-Val can also be an arithmetic expression, except when used in the Screen Section.
26. Fg-Color and Bg-Color are integer literals or numeric data items. They may be arithmetic expressions.
27. Timeout is a integer literal or numeric data item.
28. Cntrl-String is a nonnumeric literal or data item.
29. If the UPON phrase is used it must be the first optional phrase specified.
30. If the AT phrase is specified, neither the LINE nor the COLUMN phrase may be specified.

31. If the COLOR phrase is specified, neither the FOREGROUND-COLOR nor the BACKGROUND-COLOR phrase may be specified.
32. IS and "=" are synonymous.

Format 4

33. Title is an alphanumeric literal or data item
34. Save-area is a USAGE HANDLE OF WINDOW or PIC X(10) data item.

Format 5

35. Statement-1 and Statement-2 are imperative statements.

Format 6

36. Exactly one of the SIZE or LINES phrases must be specified. The selected phrase may appear anywhere in the statement.
37. The LINES phrase can take a numeric expression.

Format 8

38. Source is an alphanumeric data item or nonnumeric literal.

Format 9

39. Title-1 is an alphanumeric literal or identifier that contains the new title.
40. Handle-1 is the handle of the floating window in which the new title is applied.
41. If Handle-1 is a subwindow, then its parent window title is changed.

Format 10

42. New-Window is a USAGE HANDLE OF WINDOW or PIC X(10) data item. If used, the UPON phrase must be the first optional phrase specified
43. Font-1, font-2 and font-3 are data items described as USAGE HANDLE or HANDLE OF FONT. They should contain valid handles to screen fonts.
44. Cell-units is a positive integer data item or literal.
45. Control-type-name is one of the control type reserved words known by the compiler
46. Min-size, max-size, min-lines and max-lines are integer literals or data items.
47. The word "NO-CLOSE" is reserved by the compiler only when it appears in a Format 11 or 12 DISPLAY statement.
48. Manager is a USAGE HANDLE or HANDLE OF LAYOUT-MANAGER that contains a valid reference to a layout manager.
49. Visible-state is a numeric literal or data item.
50. Menu-1 is a USAGE HANDLE or HANDLE OF MENU data item.
51. Proc-1 and proc-2 are procedure names.
52. Action is a numeric literal or data item.
53. Height is a numeric data item or literal. It may be a non-integer value. You can also specify the value as an arithmetic expression
54. Layout is an alphanumeric literal or data item.
55. Grid-Handle is a USAGE HANDLE OF GRID data item or the screen name of a Grid control defined in the Screen Section. x and y are numeric literals or data items.

Format 11

56. The POP-UP/HANDLE phrase may be specified anywhere in the statement after the required initial elements

57. Text is a literal or data item
58. font-1 is a USAGE HANDLE OF FONT
59. Type, icon, color-val, bg-color, fg-color and default are numeric literals or data items
60. Value is a numeric data item

Format 12

61. name is an alphanumeric data item or literal that is the name of an environment variable or configuration variable

Format 13

62. Title may appear only once, either in a TITLE phrase or the initial title option. The title option must be the first option specified.

General Rules

Format 1

1. 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 literal shall be the alphanumeric representation of that figurative constant.
2. 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.
3. 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 general rules 4 and 5 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.
4. 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.
5. If the UPON phrase is not specified, the standard display device is used.
6. 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.
7. 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.
8. If vertical positioning is not applicable on the hardware device, the vertical positioning shall be ignored.

Format 2

9. Column and line number positions are specified in terms of alphanumeric character positions.
10. The DISPLAY statement causes the transfer of data in accordance with the MOVE statement rules 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 of data to the elementary screen items is done in the order that the screen items are specified within screen-name-1.

> **NOTE** - When two screen items overlap, the display on the screen for the common character positions is determined by the second screen item specified within screen-name-1.

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.

11. 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.
12. If the LINE phrase is not specified, the screen record starts on line 1.
13. If the COLUMN phrase is not specified, the screen record starts in column 1.
14. 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.
15. If the UPON clause is followed by a Tab-Control handle, the destination Tab-Control must have the Allow-Container style. When the UPON clause is followed by a Tab-Control handle or a Ribbon handle, the screen will be bind to the page specified by Page-Index and the runtime will show it when the page is selected by the user.
16. Screen-group shall reference a Screen Section group, so a Screen-name not associated to a control-class.
17. If the UPON clause is followed by a Screen-group, the content of screen-name-1 is added to the Screen identified by Screen-group like if it was declared in Screen Section at the bottom of that group

Format 3

18. The DISPLAY statement sends each of its Src-Items to the video terminal attached to the executing program. If more than one Src-Item is specified, each is treated as if it were in a separate DISPLAY statement in the order listed.
19. If the OMITTED option is used, then no Src-Item is sent to the screen. This can be used to cause the action of various optional phrases without sending any actual data, like BELL for instance. If a SIZE phrase is specified, then the OMITTED phrase will act like an alphanumeric src-item of the specified size whose value is identical to the characters located on the screen where the DISPLAY will occur. This can be used to modify the video attributes of the screen without changing the displayed data.
20. For example, DISPLAY OMITTED, SIZE 5, REVERSE will cause the five characters located at the current cursor location to be changed to reverse-video.
21. The CONTROL phrase provides the ability to modify the static attributes of the DISPLAY statement at runtime. The CONTROL data item is treated as a series of comma-separated keywords that control the action of the statement. Within the CONTROL data item, spaces are ignored and lower-case letters are treated as if they

were upper-case. The keywords allowed in cntrl-string are:

```
ERASE, ERASE EOL, ERASE EOS, NO ERASE
BEEP, NO BEEP
HIGH, LOW, STANDARD, OFF
BLINK, NO BLINK
REVERSE, NO REVERSE
TAB, NO TAB
PROMPT, NO PROMPT
CONVERT, NO CONVERT
UPDATE, NO UPDATE
ECHO, NO ECHO
UPPER, NO UPPER, LOWER, NO LOWER
UNDERLINED, NO UNDERLINE
LEFT, RIGHT, CENTERED, NO JUST
SAME
FCOLOR
BCOLOR
```

Any other keywords and spaces are discarded. If more than one keyword from within the above lines appears in cntrl-string, then only the rightmost one in the data item is used. Each of the keywords performs the same action as the statically declared attribute of the same name. When a CONTROL item conflicts with the statically declared attributes of the DISPLAY statement, the actions specified in the CONTROL item take precedence. The FCOLOR and BCOLOR keywords are used to set foreground and background colors respectively. These keywords must be followed by an equals sign and the name of a color taken from the following list: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, and WHITE. The named color becomes the default foreground or background color for the window. Note that this is different from the COLOR phrase, which sets the color only for the current DISPLAY statement. The FCOLOR and BCOLOR keywords set the default colors for every subsequent DISPLAY until explicitly changed.

22. Line and column coordinates can be specified between parenthesis soon after the DISPLAY word. This is a Microsoft COBOL extension and requires the -cms compile flag. In this case the UPON clause cannot be used.

Format 4

23. The DISPLAY SUBWINDOW verb creates or modifies the current Subwindow. The subwindow is a rectangular region of the screen. In essence, the current subwindow defines a virtual terminal screen that occupies some area of the user's physical screen. Line and column numbers for ACCEPT and DISPLAY statements are computed from the upper left-hand corner of the current subwindow. For example, the statement DISPLAY SPACE, ERASE SCREEN erases only the current subwindow.
24. When used with floating windows (Format 10), this verb creates a subwindow in the current floating window. Note that every floating window has an implicit subwindow. Each time a floating window is made current, its subwindow is also made current.
25. When used with floating windows, subwindow coordinates are relative to the current floating window. Initially, this is the main application window. The subwindow never extends past the boundaries of the current floating window.
26. The initial subwindow is set to the entire screen. When created inside a floating window, the subwindow is set to the floating window's display area (the area inside the borders, menu bar, toolbar, and title bar).
27. Any subwindows contained in a floating window are automatically closed if the floating window is closed.

ERASE Phrase

28. This phrase is implied by the BOXED and REVERSED phrases.

BOXED Phrase

29. The BOXED phrase causes a box to be drawn around the new window. The box is drawn outside of the

window. Any portions of the box that lie off the screen will not be drawn.

30. The terminal's line drawing set is used to draw the box. If the terminal does not have a line drawing set, hyphens and vertical bar characters are used.
31. If the POP-UP phrase is also specified, the box will overlay any other boxes on the screen. If this phrase is not specified, the box drawn will be attached to any other boxes it intersects.
32. This phrase implies the ERASE phrase.

REVERSED Phrase

33. The REVERSED phrase exchanges the window's foreground and background colors. This will affect every ACCEPT and DISPLAY statement in the new window.
34. This phrase implies the ERASE phrase. Note that this will usually cause the entire window to be set to reverse video spaces when it is initially created.

COLOR Phrase

35. The COLOR phrase sets the window's foreground and background colors. These colors are used whenever the window is erased and as default colors for future ACCEPT and DISPLAY statements. Color-val contains a numeric representation of the colors to use.
36. If the foreground color is not specified, the new window inherits the current window's foreground color. If the background color is not specified, the new window inherits the current window's background color.

HIGH, LOW, and STANDARD Phrases

37. The HIGH, LOW, and STANDARD phrases set the foreground intensity of the subwindow. This affects any drawing done to the subwindow itself (such as its border or title). In addition, they set the default intensity for any future ACCEPT/DISPLAY statements made to this window.

    The STANDARD option indicates that a system-dependent default value should be used. You can affect this value with the FOREGROUND-INTENSITY configuration option.

    If no option is given, the current subwindow's foreground intensity is used (inherit the foreground intensity of the parent).

BACKGROUND-HIGH, BACKGROUND-LOW, and BACKGROUND-STANDARD Phrases

38. The BACKGROUND-HIGH, BACKGROUND-LOW, and BACKGROUND-STANDARD phrases set the background intensity for the subwindow. This works in a fashion analogous to the foreground intensity described above. Note that the COLOR phrase, if present, takes precedence over the BACKGROUND phrases.

CONTROL VALUE Phrase

39. The CONTROL VALUE phrase provides a method for specifying certain window characteristics at run time. Control-value must be a numeric expression that contains one or more of the following values added together:

Boxed	1
Shadow	2
No Scroll	4
No Wrap	8
Reverse	16

TITLE Phrase

40. The TITLE phrase causes a title to be printed in the window's border. This has effect only if the BOXED phrase is also specified.
41. One top title and one bottom title may be specified for each window. Top titles can be placed in one of three positions in the border region: top left, top center, or top right. Bottom titles can be placed in the bottom left, bottom center, or bottom right. If TOP or BOTTOM is not specified, TOP is used. If LEFT, CENTERED, or RIGHT is not specified, CENTERED is used.

NO SCROLL and NO WRAP Phrases

42. Specifying NO SCROLL disables automatic scrolling for the new window. Normally, when the cursor is moved past the bottom edge of the window, the window is scrolled up one line. If NO SCROLL is specified, then the window will not be scrolled. The bottom line will be overwritten instead. When NO SCROLL is specified, then the only way to scroll a window is explicitly with a SCROLL phrase on a Format 1 DISPLAY statement.
43. Specifying NO WRAP disables line wrap for the window. Normally, a line that extends past the right edge of the window is wrapped around to the next line. If NO WRAP is specified, then the line is truncated instead. This will leave the cursor logically positioned on the same line just to the right of the window's edge. Further output will not be visible until the cursor is repositioned inside the window.
44. The scroll and wrap states of the current window are saved when a pop-up window is created. When that pop-up window is closed, the scroll and wrap states of the old window are restored.

POP-UP AREA Phrase

45. The POP-UP AREA phrase causes the screen manager to save information about the current subwindow prior to creating the new window. This information can be used by the screen manager later to remove the new window and restore the saved window. This is meant to be used to create "pop-up" windows.
46. Boxed pop-up windows are automatically detached slightly from any intersecting line segments, such as the borders of other windows.
47. The save-area is an elementary data item described by a PICTURE X(10) clause. It is filled in with information about the current window dimensions and contents before the new window is created. This data item is required for restoration of a window and must not be subsequently modified in any way. It can be referenced in a CLOSE WINDOW verb to restore the saved window to the screen and re-establish the saved window as the current window.
48. Pop-up subwindows are an older technology that is not compatible with graphical controls. You should avoid using pop-up windows if you also use controls. Use floating windows instead.

SHADOW Phrase

49. The SHADOW phrase causes the window to appear to float over the screen, giving it a three-dimensional effect.

    The way the shadow is displayed is determined by the SHADOW-STYLE setting of the SCREEN option in your runtime configuration file.

    When a shadow is specified for a window, that window is automatically detached slightly from any intersecting line segments, such as the borders of other windows.

Format 5

50. The DISPLAY SCREEN verb is used to shift between the 80-column mode and 132-column mode of the terminal. On character-based systems, the terminal is physically set to either 132- or 80-column mode. For all systems, the main application window is then set to 132 columns wide; it is cleared, and its subwindow set to cover the entire interior. Some graphical systems simulate 132-column mode by scrolling the main application window.
    A. In graphical environments, the MODIFY verb is the preferred method for changing the size of a graphical screen.

51. If the terminal hardware does not support the mode shifted to, the screen and current window do not change and the EXCEPTION phrase Statement-1 (if specified) executes. If the mode change is successful and the NOT EXCEPTION phrase is used, Statement-2 executes.

Format 6

52. The DISPLAY LINE verb provides the ability to draw horizontal and vertical lines in a machine- and terminal-independent fashion. The lines are drawn using the best mode available on the display device. Used together with the DISPLAY BOX verb, this verb provides the ability to draw forms on the user's screen.
53. If the SIZE phrase is specified, the line drawn is horizontal. The value of Length gives the size of the line in screen columns. If the LINES phrase is used instead, the line drawn is a vertical line and Height describes the number of screen rows to use.
54. Lines never wrap around or cause scrolling. If the LINES or SIZE phrase would cause the line to leave the current window, the line is truncated at the edge of the window.

AT, LINE, and COLUMN Phrases

55. The value of Line-Num gives the starting row of the line. The value of Col-Num gives the starting column. The value of Screen-Loc gives the starting row and column. Lines are always drawn to the right or downwards as appropriate. Screen-Loc, Line-Num, and Col-Num must specify a position that is contained in the current window.
56. If the LINE NUMBER phrase is not specified, line 1 is used. If the COLUMN NUMBER phrase is missing, column 1 is used.

TITLE Phrase

57. The TITLE phrase has effect only when you are drawing horizontal lines. When it is specified, Title-String is printed in part of the line.
58. The title may be printed near the right side, near the left side, or in the center of the line depending whether the RIGHT, LEFT, or CENTERED phrase is specified. If none is specified, CENTERED is used.

Format 7

59. The DISPLAY BOX verb provides the ability to draw a box in a machine- and terminal-independent manner. The best drawing mode of the display device is used. If the lines used in drawing a box intersect other lines already present on the screen, the appropriate intersection characters are used.
60. You specify the location of the box by providing the location of the upper-left corner. You specify the size of the box by providing a height and a width.

AT, LINE, and COLUMN Phrases

61. The AT, LINE NUMBER, and COLUMN NUMBER phrases operate in the same manner as they do when they are used in a DISPLAY LINE statement (Format 6).

SIZE and LINES Phrases

62. The SIZE phrase specifies the width of the box. The LINES phrase specifies its height. Length and Height must specify values greater than one. If the SIZE phrase is absent, the box will extend to the right edge of the current window. If the LINES phrase is missing, the box will extend to the bottom of the current window.

TITLE Phrase

63. The TITLE phrase operates in the same manner as it does for a DISPLAY WINDOW verb (Format 4).

Format 8

64. The DISPLAY UPON WINDOW TITLE verb is used to modify a boxed subwindow's title. Either the top or bottom title may be modified.
65. If any of the positioning phrases is used, the new title is placed in the indicated position.
66. If neither the TOP/BOTTOM nor the CENTERED/LEFT/RIGHT option is used, then the window's title is modified

in its current position. If the window has a top and bottom title, the top title is modified.

67. If the TOP/BOTTOM phrase is omitted, TOP is implied.
68. If the CENTERED/LEFT/RIGHT phrase is not used, CENTERED is implied.

Format 9

69. DISPLAY UPON FLOATING WINDOW TITLE is used to change the title of the main application window or floating window. When you are changing a floating window's title, Handle-1 identifies which window to change.
70. The case of the title will be exactly as given in Title-1.
71. The MODIFY verb can also be used to change a window's title.

Format 10

72. The syntax for DISPLAY FLOATING WINDOW is a superset of the DISPLAY WINDOW verb. This simplifies conversion of DISPLAY WINDOW statements to DISPLAY FLOATING WINDOW statements.
73. The DISPLAY FLOATING WINDOW verb creates a new floating window and stores a handle to the window in Handle-name. Use the value of Handle-name with other verbs (such as DESTROY) when you need to refer to the window.
74. After the new window is created, it becomes both the current and active window.
75. The window created may be either modal or modeless. A modal window is a window that the user cannot leave until it is closed. A modeless window is a window that the user can leave (switch to another window) while it is still open. These names are derived from the idea that a modal window enters a new mode in the program (for example, selecting a file to open) while a modeless window does not (since the user can continue working on tasks in other windows).
76. The DISPLAY FLOATING WINDOW verb also creates a new subwindow that exactly covers the interior of the floating window. This is identical to an implied DISPLAY SUBWINDOW statement (with no options). Any HIGH, LOW, STANDARD, BACKGROUND-HIGH, BACKGROUND-LOW, BACKGROUND-STANDARD, REVERSE, COLOR, NO SCROLL, or NO WRAP phrases specified in the DISPLAY FLOATING WINDOW verb are inherited by the implied subwindow.
77. Each window has a controlling thread. A window's controlling thread is the most recent thread to have created that window or done an ACCEPT from that window. When a thread performs an ACCEPT from a window, and that thread is not the controlling thread of the active window, the thread suspends. The thread remains suspended until the window it is accessing becomes active (either because the user activates it or the program does).
78. Most of the optional phrases have the same meaning as they do for DISPLAY SUBWINDOW. However, note the following exceptions:
    A. The ERASE SCREEN phrase is always implied by DISPLAY FLOATING WINDOW, so specifying this phrase has no additional effect.
    B. Most GUIs (including Windows) cannot display shadowed pop-up windows. On these systems, the SHADOW phrase has no effect. On character-based systems, the SHADOW phrase has its normal effect.
    C. All GUIs create borders around their windows. If there is a choice of border thickness, specifying BOXED will select a thicker border than omitting BOXED. Under character-based systems, the BOXED phrase determines whether or not there will be a border.
    D. The HIGH, LOW, STANDARD, BACKGROUND-HIGH, BACKGROUND-LOW, BACKGROUND-STANDARD, REVERSE, COLOR, NO WRAP, and NO SCROLL phrases do not directly affect the created window. Instead, they are passed on to the initial subwindow as described in Rule 5, above.

E. Most GUIs (including Windows) cannot display more than one window title and do not give you a choice of title position. On these systems, the specified TITLE appears in the location determined by the GUI (usually top center, or top left). If you specify more than one title, the TOP title is the one used. If you specify only one title (either TOP or BOTTOM), it is used regardless of the title location.

79. If *Handle-1* points to a subwindow, the parent window title is updated.

GRAPHICAL Phrase

80. The optional GRAPHICAL phrase directs the compiler to use a default CELL phrase equivalent to:

```
CELL SIZE = LABEL FONT
```

This phrase establishes the window's coordinate space based on the font used by controls that occupy the window. The CELL phrase can still be used and any values set in that phrase take precedence over the default value established with GRAPHICAL option. In other words, if you specify only a CELL HEIGHT or CELL WIDTH, then the other dimension receives the default assignment.

The intent of the GRAPHICAL option is to make it easier to consistently establish an appropriate coordinate space for windows that contain only controls (see the discussion of cell sizing and coordinate space that is included with the CELL phrase rules, below).

For example, the window that is specified with:

```
DISPLAY FLOATING WINDOW,
CELL SIZE = LABEL FONT
```

can be more simply specified with:

```
DISPLAY FLOATING GRAPHICAL WINDOW
```

UPON Phrase

81. The UPON phrase specifies the parent of the new floating window. Parent-window must be a valid floating window handle. If the UPON phrase is omitted, the current window is used as the parent. If you create a new floating window in the scope of an UPON phrase, the new window becomes the current window when the DISPLAY statement terminates.
82. The UPON Grid-Handle phrase allows you to create graphical controls over grid cells. Use x and y to indicate the cell coordinates.

MODAL and MODELESS Phrases

83. The word MODAL is accepted only by floating windows. Floating windows are modal by default, so this word is just commentary in the display of a floating window. When a modal window is active, all other windows are disabled. The user cannot activate another window, including any of its components (such as its menu or close button).
84. The word MODELESS makes a window modeless. All windows except for floating windows are modeless by default. When a modeless window is active, the user can activate another window by using the host system's techniques for doing so (for example, by clicking on another window with the mouse). When this happens, any ACCEPT that is active is terminated by a CMD-ACTIVATE event. Your program should respond by performing an ACCEPT in the window requested by the user. Alternatively, you can link your modeless window to a thread, see rule 1 under LINK TO THREAD and BIND TO THREAD Phrases (below).

LINK TO THREAD and BIND TO THREAD Phrases

85. The LINK TO THREAD phrase allows the runtime to automate the handling of the CMD-ACTIVATE event. If the user activates a window created with the LINK TO THREAD phrase, the runtime will examine that window to see if it has a controlling thread different from the current thread. If it does, then the current thread suspends

and the thread controlling the newly active window is allowed to run. The runtime handles all aspects of the window activation. The CMD-ACTIVATE event is not returned to the program in this case. If the controlling thread of the new window is the same as the current thread, then the runtime does not perform any special handling and the CMD-ACTIVATE event is passed on to your program. In order to get the best benefit from the LINK TO THREAD phrase, you should arrange to have a separate thread control each modeless window in your program.

86. The BIND TO THREAD phrase has the same effect as the LINK TO THREAD phrase. In addition, the window is automatically destroyed when its controlling thread terminates.

SCREEN LINE and SCREEN COLUMN Phrases

87. The SCREEN LINE and SCREEN COLUMN phrases determine the initial location of the window on the screen. Screen-line and screen-col give the coordinates of the upper left corner of the window in screen base units. Screen base units are machine dependent. On character systems, they are character cells. On graphical systems, they are pixels. The upper left corner of the screen is location "1,1". Under Windows, the runtime ensures that the initial window is fully visible, so the specified location may not be used if that would place a portion of the window off the screen (the closest allowed location is used). Windows other than the initial window may be placed arbitrarily. On graphical systems, the location of a floating window is interpreted to mean the location of its exterior. On character systems, the location is the same as it is for subwindows: the location of the window's interior.

SCREEN-INDEX Phrase

88. The SCREEN-INDEX phrase specifies the index of the monitor where the window should be displayed in a multi-monitor environment.

LINE, COLUMN, and AT Phrases

89. The LINE phrase indicates the starting row of the new window. This is always relative to the first line of the parent window. Non-integer values are allowed. If the LINE phrase is omitted, then the new window is first centered vertically over the parent window and then adjusted to be fully on the screen.
90. The COLUMN phrase works the same as the LINE phrase, except that it controls the horizontal positioning.
91. The AT phrase screen-loc item must be either a 4-digit or 6-digit number. The first half of this number is the starting row, the second half the starting column. These values are interpreted in the same manner as they are for the LINE and COLUMN phrases.

SIZE and LINES Phrases

92. The SIZE phrase indicates the width of the interior of the new window. If it is omitted, then the width is the same as the main application window. If there is no main application window available, the default size of the floating window is the same as the current window.
93. The LINES phrase indicates the height of the new window's interior. If it is omitted, then the height is the same as the main application window. As with the SIZE phrase, a non-integer number of lines may be specified. Any partial lines created are always displayed as spaces with the background color. The minimum value for LINES is "1" (one).

FONT Phrase

94. The FONT phrase assigns the font that will be used for all textual ACCEPT and DISPLAY statements used in the window. This also sets the default cell size to the size of the "0" (zero) character described by font-1. The cell size determines the height of one row and the width of one column. The font described by font-1 must be a fixed-width font. If it is not, or if the FONT phrase is not specified, then the font used is the same as the one used by the parent window.

CONTROL FONT Phrase

95. The CONTROL FONT phrase specifies the default font to use for any graphical controls displayed in this window. If you omit the CONTROL FONT phrase, a system default is used (the font "DEFAULT-FONT").

CELL Phrase

96. The CELL phrase defines the height, or width, or height and width of one cell in the window. A cell defines the height of one row and the width of one column. The default cell size is set by the size of the font used in the window.
97. The cell size is described in terms of cell units. The exact meaning of a cell unit is machine-dependent. Typically, for character-based systems, one cell unit is equal to the height or width (as appropriate) of one screen character. On graphical systems, a cell unit is typically one pixel in size. When developing programs, you should avoid writing code that depends on fixed (hard-coded) values for cell units.
98. The HEIGHT option of the CELL phrase defines the cell height for the new window. The WIDTH option defines the width. The SIZE phrase defines both the height and width together.
99. The cell-units option sets the cell's height, or width, or both, to the value of cell-units.
100. The control-type-name phrase causes the cell height, or width, or both, to be based on a particular font and control type. The system measures the size of font-2 when it is used in a control described by control-type-name, and sets the cell size accordingly. This option is typically used to set the coordinate space of the window to one that is convenient for aligning several controls of a particular type and font. Note that the font handle (font-2) is not required. When it is omitted, the window's CONTROL FONT is used. Also note that if the font handle is omitted, the optional word FONT is required (in order to avoid ambiguity with the FONT phrase).
101. If the SEPARATE option is specified, then a system-dependent amount is added to the measured font height to provide for some vertical separation between controls. This is typically used to provide some space between boxed entry-fields on adjacent rows. On the other hand, if OVERLAPPED is specified, the height is reduced by the size of the top border of a boxed entry field. This causes boxed fields on adjacent rows to share a common border.
102. The runtime currently limits control-type-name to be either a "LABEL" or "ENTRY-FIELD". If another control type name is used, the runtime treats it as if it were type "LABEL".
103. If a window's cell width does not match the width of its font, or if its cell height is less than the height of its font, then the effects of a textual ACCEPT or DISPLAY statement in that window are undefined. If its cell height is larger than its font's height, then the characters are positioned at the top of each cell, and the lower portion of the cell is filled with the text's background color.
104. In particular, if the relative size of the font you use in your controls changes in relation to the system's fixed font, then you will experience problems, including overlapping controls. This is because the default cell size that defines the coordinate space is based on a fixed-size font in order to maintain compatibility with character-based applications. The size relationship between the variable-pitch font used in controls and the default fixed-font that defines the coordinate space determines the appearance of the screen. If the relationship changes, the appearance of the screen changes. One way that this can happen is if the end user's machine is missing one of the fonts. In this case, Windows will substitute a different font, which may be a different size. To avoid these problems, define your coordinate space based on the same font that your controls use (with the CELL phrase). Then if the font changes the entire screen is rescaled uniformly.
105. For example, the following statement defines the coordinate space based on the font used with entry fields. This definition allows you to easily position entry fields vertically with "LINE 1", "LINE 2", etc., and have it look right.

```
CELL SIZE IS ENTRY-FIELD FONT SEPARATE
```

USER-GRAY, USER-WHITE, and USER-COLORS Phrases

106. The USER-GRAY, USER-WHITE, and USER-COLORS phrases provide a convenient way of matching your application's normal colors to those chosen by the user. The USER-GRAY option causes the palette manager to map color number "8" (low-intensity white) to the color that the user has chosen to use with 3-D objects on the host system. Similarly, USER-WHITE maps color number "16" (high-intensity white) to the color the user has chosen to be the normal background color for application windows. If you arrange your application so

that it uses color number "8" as the background for regions populated with graphical controls, and color number "16" for plain text regions, your application will look much like other applications on the system.

107. The USER-COLORS phrase indicates that you want to apply both the USER-GRAY and USER-WHITE options. These phrases are effective only on host graphical systems that have a palette manager. On other systems, these phrases have no effect. Also, note that the palette applies to the entire application. Because of this, you usually specify these options only on the first window you create.

TITLE-BAR Phrase

108. The TITLE-BAR phrase indicates that you want to have a title bar placed along the top edge of the new window. This phrase is automatically implied by the TITLE phrase (exception: this is not true if you also use the CONTROL VALUE phrase). Under some GUIs (including Windows), you must place a title bar in order to move the floating window with the mouse. Without a title bar, the user's ability to move the window depends on the host GUI. Note that you can have a title bar without specifying a title.

SYSTEM MENU Phrase

109. The SYSTEM MENU phrase causes a system menu (also known as a close box) to appear on the created window. This menu allows the user to close the floating window. It may also have additional properties depending on the host system. Under Windows, this menu contains the Move and Close operations. If you include a system menu, your program must be ready to act on a close window event (cmd_close) at any timeNO-CLOSE Phrase

110. The NO-CLOSE phrase causes the window's "Close" menu option to be disabled. This option can be applied only when the window is created and its effects cannot be reversed (the associated window's "Close" option is permanently disabled). The NO-CLOSE option takes precedence over other settings, including the setting of the QUIT_MODE configuration variable.

AUTO-RESIZE and RESIZABLE Phrases

111. The AUTO-RESIZE phrase specifies that the window be displayed with resizable borders. When the window is created, it is displayed full size as defined by the SIZE and LINES phrases. By dragging the resizable borders the user can reduce or increase the size of the window. The runtime automatically adds scroll bars as needed and manages any required scrolling. The window also has a maximize button that allows the user to immediately resize the window to its full size. The exact representation and functioning of the resizable borders and the maximize button is host system dependent. Although the user can change the physical size of the window, the logical size does not change. Neither do controls in the window change size or position. If AUTO-RESIZE is omitted, the window is a fixed size.

    The RESIZABLE phrase creates a window that the user can resize but omits the automatic handling provided by the AUTO-RESIZE phrase. When the user resizes the window, the size of the logical window is changed to match the new physical window. Any area that is new is displayed with spaces in the window's background color. Any area that has been removed is lost (although any permanent controls in that area will still exist). The window's subwindow is resized to fill the interior of the resized window. The subwindow's background color is changed to match the window's background color. Other traits of the subwindow remain unchanged. The program receives a NTF-RESIZED event to inform it of the new size. See the section on events for details. Windows that have the RESIZABLE attribute can use a resize layout manager to help handle the resizing and positioning of controls in the window.

    For windows with the RESIZABLE phrase, min-size and min-lines set the windows' smallest width and height respectively. This value is expressed in character cells (fractional cells are ignored). If omitted, or set to zero, the smallest window size is determined by the host system. Similarly, the max-size and max-lines values set the window's largest width and height. If omitted, or set to zero, the host system determines the largest size (usually the entire screen). For windows without the RESIZABLE phrase and for docking windows, these values are ignored.

ACTION Phrase

112. The ACTION phrase allows you to programmatically maximize, minimize, or restore a window. To use ACTION, assign it one of the following values (these names are found in isgui.def):

ACTION-MAXIMIZE	maximizes the window. It has the same effect as if the user clicked the "maximize" button. Allowed only for windows that have RESIZABLE or AUTO-RESIZE specified or implied for them.
ACTION-MINIMIZE	minimizes the window. Allowed only with INDEPENDENT windows that have the AUTO-MINIMIZE property set to true. It is not supported with other types of floating windows; if set, it is ignored by the runtime. ·ACTION-MINIMIZE has the same effect as if the user clicked the "minimize" button.
ACTION-RESTORE	If the window is currently maximized or minimized, restores the window to its previous size and position; otherwise, it has no effect. Allowed only for windows that can be maximized or minimized.

113. If you assign an ACTION value that is not allowed, then there is no effect other than to trigger the ON EXCEPTION phrase of the MODIFY statement (if present). Note that you can use the ACTION phrase to create a window that is initially maximized or minimized.

CONTROL VALUE Phrase

114. The CONTROL VALUE phrase allows you to specify certain attributes of the new window at run time instead of at compile time. Control-val must be a numeric expression. In it, you can specify certain floating window traits by adding together any of the following values:

Boxed	1
Shadow	2
No Scroll	4
No Wrap	8
Reverse	16
Title-Bar	32
System Menu	64
User-Gray	128
User-White	256

For each value specified, the corresponding attribute is given to the new window. When a value is not specified, the presence or absence of that trait depends on the other phrases included in the DISPLAY FLOATING WINDOW statement. Note that you can only give traits to a window with the CONTROL VALUE phrase; you cannot negate traits specified by the DISPLAY FLOATING WINDOW statement. For example, if you want to specify at run time whether or not a window gets a shadow, you should omit the SHADOW phrase from the DISPLAY FLOATING WINDOW statement and use a CONTROL VALUE phrase to add shadowing when you want it.

LAYOUT-MANAGER Phrase

115. The LAYOUT-MANAGER phrase attaches a layout manager to the window.

VISIBLE Phrase

116. The VISIBLE option determines whether the window created is visible or invisible. If the FALSE option is used, or visible-state is the value zero, then the window is invisible. Otherwise, the window is visible. If the VISIBLE phrase is omitted, then the window is visible.

POP-UP MENU Phrase

117. The POP-UP MENU phrase associates a pop-up menu with the window. If menu-1 is specified, then the menu associated with menu-1 becomes the pop-up menu. If NULL is specified, the window is not given a pop-up menu. Pop-up menus are activated by a machine-dependent technique. Under Windows, the technique is to right-click on the window's background.

CONTROLS-UNCROPPED Phrase

118. Normally, when you create a control in a window, the control is cropped to fit the current subwindow's dimensions. In addition, if the control's home position is outside of the current subwindow, the control is not created. Adding the phrase CONTROLS-UNCROPPED overrides these rules. When this phrase is used, the control is created with the specified location and dimensions, regardless of whether the control will be physically in the window.

    This can be useful when you are dealing with RESIZABLE windows. Sometimes a resizable window is too small to show all of the controls that your program creates. Normally, these controls either would not be created or would be cropped. This could produce odd results when the window is later resized larger by the user. Although the resized window is now large enough to show everything, the controls still show their cropped appearance, because their (cropped) creation size is recorded in the controls as their actual size. Specifying CONTROLS-UNCROPPED avoids the cropping behavior.

    This style is useful also when you want to place a combo-box near the bottom of a window. Because the size of the drop-down portion of the combo-box is determined by the control's overall height, cropping the control limits the drop-down box to the window's boundaries. If you want the box to drop down beyond the edge of the window, you need to use the CONTROLS-UNCROPPED window style to allow this.

EVENT PROCEDURE Phrase

119. A window's event procedure is executed whenever an event is processed for that window. The event procedure is executed as if it were the target of a PERFORM statement. Only the window's own events trigger the event procedure. Events generated by controls contained in the window do not trigger the window's event procedure (they trigger the control's event procedure instead). The event procedure executes while the event is being processed, before the event causes termination of any executing ACCEPT statement.

120. Specifying proc-1 assigns that procedure as the window's event procedure. Flow of control returns at the end of proc-1, unless proc-2 is specified, in which case flow of control returns at the end of proc-2. If you specify the NULL option, the window does not have an event procedure. This is the default, so the NULL option is treated as commentary

121. DISPLAY INITIAL WINDOW verb creates the main application window. The main application window has several special properties. If it is minimized, all other windows in the application are also minimized. If it is closed, the application terminates. A program can have only one main application window.

    If you attempt to create a main application window after one already exists, the DISPLAY INITIAL WINDOW statement will have no effect other than to set handle-name to NULL.

    The runtime automatically constructs the main application window if needed. This occurs any time a screen operation is dictated by the program and the program has not yet constructed a main application window. When this occurs, the runtime executes the following implied statement:

    A. DISPLAY INITIAL WINDOW

    B. TITLE-BAR,

C. SYSTEM MENU,

D. AUTO-MINIMIZE,

E. AUTO-RESIZE.

122. The main application window is always modeless. A modeless window is one where the user can switch to another window while the current window is still open. You can include the word MODELESS in the statement as commentary.

123. The INDEPENDENT phrase creates an independent window. Independent windows act like additional main application windows. Independent windows have the following traits:

Independent windows do not have a parent. As a result, any other window in the application can be placed over them. Also, destroying another window in the application will not destroy the independent window.

Although they do not have a parent, independent windows use the current window to determine their default fonts, cell size, and colors. Also, independent windows use the current window when determining their position. This is computed in the same manner as it is for floating windows.

Independent windows can be minimized separately. Under Windows and Windows NT, each visible independent window has its own button on the task bar.

Independent windows process their close box in the same manner as floating windows--by generating a CMD-CLOSE event.

Independent windows can be created before the main window. In this case, there is no current window to provide defaults, so the independent window uses the same defaults as the main application window would.

If an independent window is current when the main application window is created, the defaults for the main window are derived from the independent window.

124. Most of the phrases allowed for DISPLAY INITIAL WINDOW work in exactly the same way that they work in a DISPLAY FLOATING WINDOW (format 11) statement. The following rules are supplemental.

SCREEN-LINE and SCREEN-COLUMN Phrases

125. Under Windows, the runtime ensures that the initial window is fully visible, so the location specified by screen-line and screen-col may not be used if that would place a portion of the window off the screen (the closest allowed location is used).

AUTO-MINIMIZE Phrase

126. The AUTO-MINIMIZE phrase indicates that a minimize button should be displayed. The runtime handles the minimizing and restoring of the application automatically. If you do not specify AUTO-MINIMIZE, the user is not allowed to minimize the application.

In addition, the AUTO-MINIMIZE phrase implies the SYSTEM MENU and TITLE-BAR phrases.

UNDECORATED Phrase

127. The UNDECORATED phrase turns off native decorations like frame and title bar.

STANDARD Phrase

128. The STANDARD option is identical to the INITIAL option except that it automatically implies the following options:

A. TITLE-BAR

B. SYSTEM MENU

C. AUTO-MINIMIZE

D. USER-COLORS

129. For graphical systems, a black foreground on a white background. For character-based systems, a white foreground on a black background. You may override these default colors with the various color setting phrases.

130. DISPLAY TOOL-BAR adds a toolbar to the current floating or main-application window. A toolbar is a region of the window that is devoted to holding buttons and other controls that are used as mouse accelerators that the user can activate with the mouse to quickly specify a program operation. Toolbars extend across the entire width of the window. They appear at either the top or the bottom of the window depending on the host system. Under Microsoft Windows, toolbars appear at the top of the window.

131. You may have more than one toolbar associated with a particular window. All toolbars associated with a window are stacked vertically in the order that they are created.

132. Like menu-bars, the space that a toolbar occupies is not part of the body (client area) of the window. When you create a toolbar, the window it is attached to is enlarged to accommodate the space required by the toolbar unless the window is RESIZABLE. Toolbars automatically grow and shrink horizontally to match the width of the owning window.

133. In several respects, toolbars act like windows. Because of this, they are called child windows of the window they are attached to. When you create a toolbar, a handle that identifies it is returned in handle-name. Like other window handles, you can use handle-name in an UPON phrase of a DISPLAY statement to direct output to the toolbar. You can also use handle-name in a DESTROY statement to remove the toolbar. Unlike other windows, toolbars are not made current or active when they are created. This means that the only way to place a control on the toolbar is with the UPON phrase.

134. Toolbars cannot take textual output (i.e., text-style ACCEPT and DISPLAY statements cannot refer to a toolbar). You can only place graphical controls on a toolbar. However, any type of graphical control may be placed on a toolbar. Most commonly, push buttons appear on toolbars (either text or bitmap buttons), as well as bitmap check-boxes and radio-buttons.

135. Font-1 identifies the default font to use for any controls shown in the toolbar. If font-1 is not specified, DEFAULT-FONT is used.

136. Height specifies the height of the toolbar. Height units are derived from the height of font-1 (or DEFAULT-FONT if font-1 is not used). If height is omitted, then the toolbar uses a host-dependent default height.

137. The toolbar directly uses the background-color and background-intensity only when it is drawn. The foreground-color and intensity are used as defaults for controls displayed on the toolbar. Any color and intensity elements that are omitted from the DISPLAY TOOL-BAR statement are inherited from the window that the toolbar is attached to.

138. It's good practice to populate a toolbar with controls that can act like function keys. In this way, the toolbar acts in a manner that is very similar to a menu bar. This simplifies other programming because you do not need to attempt to directly activate the controls on the toolbar. Controls that can act like function keys are buttons (push buttons, check boxes and radio buttons). Use the SELF-ACT style for buttons placed on a toolbar. For check boxes and radio buttons, use both the SELF-ACT and NOTIFY styles.

Format 11

139. Format 11 (DISPLAY MESSAGE BOX) creates a modal pop-up window with a title-bar, a text message, an icon (where available) and user defined push buttons. It then waits for the user to push one of the buttons and returns the results at which point the window is destroyed. Message boxes come in "OK" and "Yes/No" formats, with an optional "Cancel" button in each format. Message boxes are a programming convenience when you need to create a simple dialog box that can fit one of these predefined formats.

    Text forms the body of the message box. It is the string of text that the user will see. When more than one text item is specified, they are concatenated together to form a single string that the user sees.
    The message string is split across multiple lines in the box. The maximum line length is the space in pixels occupied by 53 occurrences of the character "0" using the message box font.

When the line length is reached:

- if there is no space before the line length limit, then the runtime breaks the text and shows the exceeding part on the next line,
- if there's a space before the line length limit, then this space is replaced by a line feed so the text after the space goes on the next line.

If you want to force the text to a new line, you can embed an ASCII line-feed character (h"0A" in COBOL) where you want the new line to start. If you want to insert a tabulation, you can embed an ASCII tab character (h"09" in COBOL). For example, the following code produces a message box with two lines of text, with the second line indented by one tabulation:

```
78  NEWLINE      VALUE H"0A".
78  TABULATION   VALUE H"09".
...
DISPLAY MESSAGE BOX,
"This is line 1", NEWLINE,

TABULATION "and this is line 2" .
```

140. When text is numeric, it is converted to a text string using the CONVERT phrase rules. Leading spaces in the resulting string are suppressed in the message. When text is numeric-edited, leading spaces are suppressed in the message, and the rest of the item is displayed without modification. For all other data types, text is displayed without modification.
141. Title is displayed in the message box's title bar. If title is omitted, the message box displays the same title as the application's main window.
142. Type, icon, default and value use a set of constants to describe the type of message box and the buttons it contains. These constants have level 78 definitions for them in the COPY library "isgui.def" supplied with the

compiler. These constants are as follows:

```
78  MB-OK                    VALUE 1.
78  MB-YES-NO                VALUE 2.
78  MB-OK-CANCEL             VALUE 3.
78  MB-YES-NO-CANCEL         VALUE 4.
78  MB-RETRY-CANCEL          VALUE 5.
78  MB-ABORT-RETRY-IGNORE    VALUE 6.
78  MB-CANCEL-RETRY-CONTINUE VALUE 7.

78  MB-YES                   VALUE 1.
78  MB-NO                    VALUE 2.
78  MB-CANCEL                VALUE 3.
78  MB-ABORT                 VALUE 4.
78  MB-RETRY                 VALUE 5.
78  MB-IGNORE                VALUE 6.
78  MB-CONTINUE              VALUE 7.

78  MB-DEFAULT-ICON          VALUE 1.
78  MB-WARNING-ICON          VALUE 2.
78  MB-ERROR-ICON            VALUE 3.
```

A. Type describes the set of buttons contained in the box. The possible values are:

Type Value	Buttons
MB-OK	"OK" button
MB-YES-NO	"Yes" and "No" buttons
MB-OK-CANCEL	"OK" and "Cancel" buttons
MB-YES-NO-CANCEL	"Yes", "No" and "Cancel" buttons
MB-RETRY-CANCEL	"Retry" and "Cancel" buttons
MB-ABORT-RETRY-IGNORE	"Abort", "Retry" and "Ignore" buttons
MB-CANCEL-RETRY-CONTINUE	"Cancel", "Retry" and "Continue" buttons

   i. If type is omitted, or if it contains an invalid value, then MB-OK is used.

B. Icon describes the icon that will appear. The icon appears only under Windows. On other systems, the icon selected is ignored. If icon is set to MB-ERROR-ICON, then a "stop" icon is shown. If icon is set to MB-WARNING-ICON, then an "exclamation" icon displays. If icon is set to MB-DEFAULT-ICON, then boxes with "OK" buttons will display an "information" icon, and boxes with "Yes/No" buttons will display a "question mark" icon. If icon is omitted or contains an invalid value, then MB-DEFAULT-ICON is used. Icons are customizable by providing custom GIF files as described in Default icons.

C. Font-1 describes the font used to rendered the text in the message box. If omitted, the font specified by iscobol.gui.messagebox.font is used.

D. Color-val, bg-color and fg-color describe the colors used in the dialog. If omitted, the colors specified by iscobol.gui.messagebox.bcolor and iscobol.gui.messagebox.fcolor are used.

E. Default describes which button will be the default button (i.e., the button used if the user simply presses "return"). The possible values are:

Value	Buttons
MB-OK	"OK" button
MB-YES	"Yes" buttons
MB-NO	"No" button
MB-CANCEL	"Cancel" button
MB-ABORT	"Abort" button
MB-RETRY	"Retry" button
MB-IGNORE	"Ignore" button
MB-CONTINUE	"Continue" button

i. If default is omitted, or contains an invalid value, then the default button will be the first button.

F. Timeout is the number of hundreds of seconds to wait for user input. Once this timeout is expired, the message box dialog is automatically closed. If Timeout is omitted, then the message box dialog stays active until the user performs some action.

G. The CENTERED clause causes the message box dialog to be centered in the current screen. Without this clause, the message box dialog is centered in the parent window or, if there is no parent window, it's displayed in the top left corner of the current screen. The CENTERED clause can be also activated via the iscobol.gui.messagebox.centered (boolean) configuration property.

143. Value will contain the identity of the button the user pressed to leave the message box. It uses the same values as default does, described above. For example, if the user presses the "No" button, then value will contain MB-NO.

144. On Windows systems, the system sound associated to the message type is played when the message box is displayed.

Custom message box implementation

145. It's possible to provide a custom implementation of the message box dialog through a COBOL program with

the following structure:

```
       PROGRAM-ID. MyMessageBox.

       CONFIGURATION SECTION.

       INPUT-OUTPUT SECTION.

       FILE SECTION.

       WORKING-STORAGE SECTION.
       77  rc                      pic s9.

       LINKAGE SECTION.

       77  msgbox-text              pic x any length.
       77  msgbox-title             pic x any length.
       77  msgbox-type              pic 9.
       77  msgbox-icon              pic 9(5).
       77  msgbox-default-btn       pic 9.
       77  msgbox-timeout           pic 9(5).
       77  msgbox-centered          pic 9.
           88 is-centered           value 1 false 0.


       PROCEDURE DIVISION USING  msgbox-text
                                 msgbox-title
                                 msgbox-type
                                 msgbox-icon
                                 msgbox-default-btn
                                 msgbox-timeout
                                 msgbox-centered
                                 .

       MAIN.
      *---------------------------------------------------------------*
      *                    your logic here                            *
      *---------------------------------------------------------------*
           GOBACK rc.
```

The program receives the message box specifications as Linkage parameters and can return a numeric exit code, that is useful to communicate the user choice for yes-no messages.
To associate the COBOL program to the message box implementation, use the iscobol.gui.messagebox.custom_prog configuration property, e.g.

```
iscobol.gui.messagebox.custom_prog=MYMESSAGEBOX
```

The program will be searched for either in the Classpath or in the code-prefix on the server machine, like a standard COBOL program that gets called through a CALL statement.
If you prefer to have the program searched for either in the Classpath or in the code-prefix on the client machine, like a standard COBOL program that gets called through a CALL CLIENT statement, set the property as follows:

```
iscobol.gui.messagebox.custom_prog=MYMESSAGEBOX,C
```

Format 12

146. DISPLAY UPON ENVIRONMENT-NAME sets ENVIRONMENT-NAME to the variable or Runtime Framework property setting specified by name. The value of the variable identified by ENVIRONMENT-NAME can be queried with a Format 8 ACCEPT statement.

147. DISPLAY UPON ENVIRONMENT-VALUE sets the value of the variable identified by ENVIRONMENT-NAME. This value can be queried with a Format 8 ACCEPT statement.

148. DISPLAY UPON ENVIRONMENT-VALUE fails if ENVIRONMENT-NAME hosts the name of a configuration variable whose value cannot be changed. In this case Statement-1 (if specified) is executed. Otherwise, Statement-2 (if specified) is executed.

Format 13

149. A Format 13 DISPLAY creates a new control and displays it either on the screen or within a GRID cell depending on the UPON phrase. A handle to the new control is returned in `Control-Handle` and is used in future references to the control.

150. `control-class` identifies the type of the control. See Controls Reference for the list of available controls.

151. `y` and `x` are respectively the row and column coordinates that identify a GRID cell.

152. `Screen-group` shall reference a Screen Section group, so a Screen-name not associated to a control-class.

153. IIf the UPON clause is not specified, the control is added to the current Screen but not included in the Accept of the Screen. It must be accepted separately. If the UPON clause followed by a `screen-group` is specified, the control is added to the Screen identified by `Screen-group` as if it was declared in Screen Section at the bottom of that group.

Format 14

154. Since this statement causes the creation of a pop-up window, the CONTROL is supported only for

documentative purposes. The *wcb* data item should be defined as follows:

```
01  WCB.
    03 WCB-HANDLE PIC 999 BINARY(2) VALUE 0.
    03 WCB-NUM-ROWS PIC 999 BINARY(2).
    03 WCB-NUM-COLS PIC 999 BINARY(2).
    03 WCB-LOCATION-REFERENCE PIC X.
       88 WCB-SCREEN-RELATIVE VALUE "S".
       88 WCB-WINDOW-RELATIVE VALUE "W".
    03 WCB-BORDER-SWITCH PIC X.
       88 WCB-BORDER-ON VALUE "Y" FALSE "N".
    03 WCB-BORDER-TYPE PIC 9.
       88 WCB-BORDER-WCB-CHAR VALUE 0.
       88 WCB-BORDER-PLUS-MINUS-BAR VALUE 1.
       88 WCB-BORDER-LINE-DRAW VALUE 2.
       88 WCB-BORDER-DBL-LINE-DRAW VALUE 3.
    03 WCB-BORDER-CHAR PIC X.
    03 WCB-FILL-SWITCH PIC X.
       88 WCB-FILL-ON VALUE "Y" FALSE "N".
    03 WCB-FILL-CHAR PIC X.
    03 WCB-TITLE-LOCATION PIC X.
       88 WCB-TITLE-TOP VALUE "T".
       88 WCB-TITLE-BOTTOM VALUE "B".
    03 WCB-TITLE-JUSTIFICATION PIC X.
       88 WCB-TITLE-CENTER VALUE "C".
       88 WCB-TITLE-LEFT VALUE "L".
       88 WCB-TITLE-RIGHT VALUE "R".
    03 WCB-TITLE-LENGTH PIC 999 BINARY(2).
       88 WCB-TITLE-LENGTH-COMPUTE VALUE 0.
    03 WCB-TITLE PIC X(40).
```

The border is always drawn in the same way and fill characters are not supported, hence the variables WCB-BORDER-SWITCH, WCB-BORDER-TYPE, WCB-BORDER-CHAR, WCB-FILL-SWITCH and WCB-FILL-CHAR are ignored.

The pop-up window created by this statement must be closed with a Format 3 CLOSE statement.

Format 15

155. Notification windows are undecorated dialogs that host SCREEN SECTION items. The items are displayed over the window with a Format 2 DISPLAY statement. Only labels and bitmaps should be displayed on a notification window.

156. The window appears in the position specified by the styles TOP, BOTTOM, RIGHT and LEFT. These styles can be used alone or combined together. For example, in order to display the notification window in the top-right corner of the screen, use both TOP and RIGHT.

157. ACCEPT can't be performed on notification windows . The screen hosted by a notification window never has the focus and accepting that screen would make the ACCEPT terminate with CRT STATUS set to 97 (no input fields). However, it's possible to display SELF-ACT buttons on a notification window; when these buttons are clicked, their EXCEPTION-VALUE is intercepted by the ACCEPT on the active window. A notification window is automatically closed when the user clicks on it or when `Timeout` expires.

158. If the notification window is displayed as hidden (with the VISIBLE attribute set to false) and made visible later (modifying the VISIBLE attribute to true), then a fade-in effect is applied. A fade-out effect is always applied when the window is destroyed.

Examples

Format 1 - Display to sysout where no GUI terminal exist

```
display "Hello World!" upon sysout
```

Format 2 - Display screen from screen section

```
       working-storage section.
       01 window-handle usage handle.
       01 cust-values.
          05 ws-cust-code  pic x(5).
          05 ws-cust-name  pic x(50).

       screen section.
       01 screen-1.
          03 scr-cust-code Entry-Field
             using ws-cust-code
             line 3.0
             column 21.4
             size 12.7 cells
             lines 3.8 cells
             id 1
             3-d.
          03 scr-cust-name Entry-Field
             using ws-cust-name
             line 8.7
             column 21.4
             size 24.5 cells
             lines 4.6 cells
             id 2
             3-d.
          03 scr-lab-1 Label
             line 2.7
             column 4.5
             size 13.5 cells
             lines 3.7 cells
             id 3
             title "Code :".
          03 scr-lab-2 Label
             line 9.3
             column 4.5
             size 13.5 cells
             lines 3.7 cells
             id 4
             title "Name :".
          03 scr-pb-save Push-Button
             line 16.4
             column 15.7
             size 15.5 cells
             lines 6.4 cells
             id 5
             title "Save".
         ...
```

```
       procedure division.
       display-and-accept.
          display standard  window background-low
              screen line 41
              screen column 91
              size 49.7
              lines 24.9
              cell width 10
              cell height 10
              label-offset 20
              color 257
              modeless
              title "Customers"
              handle window-handle.
           display screen-1.
           perform until exit-pushed
              accept screen-1 on exception
                 perform is-screen-1-evaluate-func
              end-accept
           end-perform.
           destroy window-handle.
       ...
```

Format 3 - Display variables and literals on specific line and column

```
display "Customer Code : " line 2 col 10
         cust-code         line 2 col 30
        "Customer Name : " line 3 col 10
         cust-full-name    line 3 col 30
end-display
```

Format 3 - Display variables and literals with Microsoft COBOL compliant syntax

```
display (2,10) "Customer Code : " cust-code
display (3,10) "Customer Name : " cust-last-name ", " cust-first-name
```

Format 4 - Display text window with some contents and title

```
display window erase
display "Press Enter to Exit" upon window bottom centered title
display "This is a help text..." line 4 col 5
display "This is a help text..." line 5 col 5
accept omitted
```

Format 5 - Change screen size to 132 on text only devices

```
display screen size 132
```

Format 6 - Display lines on text only screen

```
display line size 80 at line 2  col 1 centered title "Customers screen"
display line size 80 at line 23 col 1
```

Format 7 - Display box on text only screen

```
display box at line 2 size 80 lines 22
```

Format 8 - Display text window with some contents and title

```
display window erase
display "Press Enter to Exit" upon window bottom centered title
display "This is a help text..." line 4 col 5
display "This is a help text..." line 5 col 5
accept omitted
```

Format 9 - Display title upon floating window

```
display "Select a customer from list" upon floating window sub-win-handle title
```

Format 10 - Display graphical window and display/accept a screen on it

```
      working-storage section.
       01 window-handle usage handle.
       01 cust-values.
          05 ws-cust-code  pic x(5).
          05 ws-cust-name  pic x(50).

       screen section.
       01 screen-1.
          03 scr-cust-code Entry-Field
             using ws-cust-code
             line 3.0
             column 21.4
             size 12.7 cells
             lines 3.8 cells
             id 1
             3-d.
          03 scr-cust-name Entry-Field
             using ws-cust-name
             line 8.7
             column 21.4
             size 24.5 cells
             lines 4.6 cells
             id 2
             3-d.
          03 scr-lab-1 Label
             line 2.7
             column 4.5
             size 13.5 cells
             lines 3.7 cells
             id 3
             title "Code :".
          03 scr-lab-2 Label
             line 9.3
             column 4.5
             size 13.5 cells
             lines 3.7 cells
             id 4
             title "Name :".
          03 scr-pb-save Push-Button
             line 16.4
             column 15.7
             size 15.5 cells
             lines 6.4 cells
             id 5
             title "Save".
        ...
```

```
       procedure division.
       display-and-accept.
          display standard  window background-low
              screen line 41
              screen column 91
              size 49.7
              lines 24.9
              cell width 10
              cell height 10
              label-offset 20
              color 257
              modeless
              title "Customers"
              handle window-handle.
           display screen-1.
           perform until exit-pushed
              accept screen-1 on exception
                 perform is-screen-1-evaluate-func
              end-accept
           end-perform.
           destroy window-handle.
       ...
```

Format 11 - Display message box to request an answer

```
display message "Please confirm the deletion of this invoice"
   type mb-yes-no
   giving delete-answer
end-display
if delete-answer = mb-yes
   delete customers
else
   display message "Delete action cancelled"
end-if
```

Format 12 - Display upon environment-name

```
display "envfullname" upon environment-name
display "Adam Smith" upon environment-value
accept f-name from environment-value
```

Format 13 - Display combo-box upon grid cell, row 2, col 1

```
display combo-box upon my-grid-handle (2,1)
        handle in my-combo-box-handle
end-display
```

Format 13 - Create a Screen with 2 entry-fields and dynamically add a OK button to it

```
       WORKING-STORAGE SECTION.
       77 w-user pic x any length.
       77 w-pwd  pic x any length.
       77 ok-btn handle of push-button.

       SCREEN SECTION.
       01 screen-1.
          03 entry-field line 2, col 2, size 10 cells, using w-user.
          03 entry-field line 4, col 2, size 10 cells, using w-pwd.

       PROCEDURE DIVISION.
       ...
           display screen-1.
           display push-button ok-button line 6, col 2
                   upon screen-1
                   handle ok-btn.
```

Format 15 - Display a “job done” notification on the bottom left of the screen

```
working-storage section.
...
77 n-win handle of window.
...
screen section.
...
01 job-complete-scr.
   03 label "Job completed" line 2, col 2.
...
procedure division.
...
display notification window
        background-color rgb x#ffffff
        lines 4, size 40
        bottom right
        before time 200
        handle n-win.
display job-complete-scr upon n-win.
```

DIVIDE

Format 1

```
DIVIDE {Identifier-1} INTO { Identifier-2 [ROUNDED] } ...
       {Literal-1   }
       {Class-1     }

  [ ON SIZE ERROR Imperative-Statement-1 ]

  [ NOT ON SIZE ERROR Imperative-Statement-2 ]

  [END-DIVIDE]
```

Format 2

```
DIVIDE {Identifier-1} INTO {Identifier-2}
       {Literal-1   }      {Literal-2   }
       {Class-1     }      {Class-2     }

  GIVING { Identifier-3 [ROUNDED] } ...

  [ ON SIZE ERROR Imperative-Statement-1 ]

  [ NOT ON SIZE ERROR Imperative-Statement-2 ]

  [END-DIVIDE]
```

Format 3

```
DIVIDE {Identifier-2} BY {Identifier-1}
       {Literal-2   }    {Literal-1   }
       {Class-2     }    {Class-1     }

  GIVING { Identifier-3 [ROUNDED] } ...

  [ ON SIZE ERROR Imperative-Statement-1 ]

  [ NOT ON SIZE ERROR Imperative-Statement-2 ]

  [END-DIVIDE]
```

Format 4

```
DIVIDE {Identifier-1} INTO {Identifier-2}
       {Literal-1   }      {Literal-2   }
       {Class-1     }      {Class-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

```
DIVIDE {Identifier-2} BY {Identifier-1}
       {Literal-2   }    {Literal-1   }
       {Class-2     }    {Class-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. Identifier-1 and identifier-2 shall reference an elementary data item of category numeric.
2. Identifier-3 and identifier-4 shall reference an elementary data item of category numeric or numeric-edited.
3. Literal-1 and literal-2 shall be numeric literals.
4. Class-1 and Class-2 shall be OBJECT-REFERENCE of Java numeric primitive types (i.e. "int") or objects (i.e. "java.lang.Integer")
5. When native arithmetic is in effect, the composite of operands described in 14.6.6, Arithmetic statements, is determined by using all of the operands in the statement excluding the data item that follows the word REMAINDER.

General rules

All Formats

1. The process of determining the dividend and determining the divisor consists of the following:
   A. The dividend is identifier-2 or literal-2. The divisor is identifier-1 or literal-1.
   B. If an identifier is specified, item identification is done and the content of the resulting data item is the dividend or divisor.
   C. If a literal is specified, the value of the literal is the dividend or divisor.

Format 1

2. The evaluation proceeds in the following order:
   A. The initial evaluation consists of determining the divisor.
   B. This divisor is used with each dividend, which is each identifier-2 proceeding from left to right. Item identification for identifier-2 is done as each dividend is determined. The quotient is then formed as specified in general rule 1 and stored in the corresponding identifier-2.

Formats 2 and 3

3. The evaluation proceeds in the following order:
   A. The initial evaluation is determining the divisor and determining the dividend.
   B. The quotient is then formed as specified in general rule 1 and stored in each identifier-3.

Formats 4 and 5

4. The evaluation proceeds in the following order:
   A. The initial evaluation is determining the divisor and determining the dividend.
   B. The quotient is then formed as specified in general rule 1 and stored in identifier-3.
   C. If the size error condition is not raised, a subsidiary quotient is developed that is signed and derived from the quotient by truncation of digits at the least significant end and that has the same number of digits and the same decimal point location as the data item referenced by identifier-3. The remainder is stored in the data item referenced by identifier-4 unless storing the value would cause a size error condition, in which case the content of identifier-4 is unchanged.

Examples

Format 1 - Divide literal into variable and validate size error.

```
move    10500   to num-years
divide 365.25 into num-years
    on size error display message "Invalid result size of last divide operation!"
    not on size error display message "Number of years : " num-years
end-divide
```

Format 2 - Divide variable into variable and leave result in separate variable, validating size error.

```
move 365 to year-days
move  15 to period-days
move zero to num-periods-year
divide period-days into year-days giving num-periods-year
 on size error display message "Invalid result size of last divide operation!"
 not on size error display message "Number of periods a year : " num-periods-year
end-divide
```

Format 3 - Divide variable by variable and leave result in separate variable, validating size error.

```
move 365 to year-days
move  15 to period-days
move zero to num-periods-year
divide year-days by period-days giving num-periods-year
 on size error display message "Invalid result size of last divide operation!"
 not on size error display message "Number of periods a year : " num-periods-year
end-divide
```

Format 4 - Divide variable into variable leaving result in separate variable and remainder in another variable too.

```
move 365 to year-days
move  15 to period-days
move zero to num-periods-year
divide period-days into year-days giving num-periods-year
       remainder not-used-days
display message
        "A year of 365 days, has " num-periods-year
         " periods of 15 days plus " not-used-days " days"
```

Format 5- Divide variable by variable leaving result in separate variable and remainder in another variable too.

```
move 365 to year-days
move  15 to period-days
move zero to num-periods-year
divide year-days by period-days giving num-periods-year
       remainder not-used-days
display message
        "A year of 365 days, has " num-periods-year
         " periods of 15 days plus " not-used-days " days"
```

ENTER

General Format

```
ENTER language-name [routine-name].
```

Syntax Rules

1. Language-name and routine-name can be any user-defined word or alphanumeric literal.

General Rules

1. This statement is treated as for documentation purposes only. Access to other languages can be achieved by means of the CALL statement.

ENTRY

General Format

```
ENTRY Entry-Name [ USING [ BY { VALUE     } ]{ Identifier } ... ]
                              { REFERENCE }
```

Syntax rules

1. *Entry-Name* is a non-numeric literal.
2. *Identifier* is a data item defined with a level-number of 01 or 77 and it must appear in the Linkage Section.
3. Each *Identifier* cannot appear more than once in the USING phrase.

General rules

1. The Linkage Section is referenced for parameters in the USING phrase.
2. The ENTRY point must be loaded and present in memory before a CALL statement referencing the ENTRY point is used. Once loaded, COBOL programs may be called by any of their ENTRY point names. Name matching logic is not case sensitive.
3. Both the BY REFERENCE and the BY VALUE phrases are transitive across the parameters that follow them until another BY REFERENCE or BY VALUE phrase is encountered. If neither the BY REFERENCE nor the BY VALUE phrase is specified prior to the first parameter, the BY REFERENCE phrase is assumed.

Examples

Program with several entry points and program to call the different entry-points

```
program-id. testentry.

linkage section.
01 par-1  pic x(5).
01 par-2  pic x(5).
01 par-3  pic 9(5).

procedure division.
main.
 perform finish.
 entry "tentry1" using par-1
    perform p-entry1.
 entry "tentry2" using par-2
    perform p-entry2.
 entry "tentry3" using par-1 par-3
    perform p-entry3.

finish.
  goback.

p-entry1.
  display message "Entry 1 : " par-1
  perform finish.

p-entry2.
  display message "Entry 2 : " par-2
  perform finish.

p-entry3.
  display message "Entry 3 : " par-1 ", " par-3
  perform finish.
```

```
program-id. callentry.

working-storage section.
01 ws-par-1  pic x(5) value "hello".
01 ws-par-2  pic x(5) value "bye".
01 ws-par-3  pic 9(5) value 12345.

procedure division.
main.
  call "testentry"
  call "tentry3" using ws-par-1 ws-par-3
  call "tentry2" using ws-par-2
  call "tentry1" using ws-par-1
  goback.
```

EVALUATE

General Format

```
EVALUATE  {Sel-Subject} { [ ALSO {sel-subject} ] } ...

  { { WHEN Sel-Object { [ ALSO Sel-Object ] } ... } ... Statement-1 } ...

  [ WHEN OTHER Statement-2 ]

  [END-EVALUATE]
```

Where *Sel-Subject* is:

```
   { Identifier-1                          }
   { Literal-1                             }
   { Arithmetic-Expression-1               }
   { Boolean-Expression-1                  }
   { Condition-1                           }
   { TRUE                                  }
   { FALSE                                 }
```

Where *Sel-Object* is:

```
   { [IS NOT] Identifier-2                   }
   { [IS NOT] Literal-2                      }
   { [IS NOT] Arithmetic-Expression-2        }
   { [IS NOT] Comparison-Expression          }
   { [IS NOT] Boolean-Expression-2           }
   { [IS NOT] Range-Expression               }
   { Condition-2                           }
   { Partial-Expression-1                  }
   { TRUE                                  }
   { FALSE                                 }
   { ANY                                   }
```

Where *Range-Expression* is:

```
   { Identifier-3            } {THROUGH} { Identifier-4            }
   { Literal-3               } {THRU   } { Literal-4               }
   { Arithmetic-Expression-3 }           { Arithmetic-Expression-4 }
```

Syntax rules

1. The words THROUGH and THRU are equivalent.
2. The number of selection objects within each set of selection objects shall be equal to the number of selection subjects.
3. The two operands in a range-expression shall be of the same class and shall not be of class boolean, object or pointer.
4. A selection object is a partial-expression if the leftmost portion of the selection object is a relational operator, a class condition without the identifier, a sign condition without the identifier, or a sign condition without the arithmetic expression.
5. The classification of some selection subjects or selection objects is changed 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 and therefore condition-2.

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 and therefore condition-1.

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 and therefore boolean-expression-2.

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 and therefore boolean-expression-1.

E. If the selection object is 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.

6. 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 in accordance with 8.8.4.1.1,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 word ANY may correspond to a selection subject of any type.

   D. Partial-expression-1 shall correspond to a selection subject that is an identifier, a literal, an arithmetic expression, or a boolean 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.

7. 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 preceding partial-expression-1 by the selection subject. The corresponding selection subject is treated as though it were specified by the word TRUE.

8. The permissible combinations of selection subject and selection object operands are indicated in the table below

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] comparison-expression	Y	Y	Y	Y		
[NOT] boolean-expression	Y	Y		Y		
[NOT] range-expression	Y	Y	Y			
Condition					Y	Y

Partial-expression	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 an operand of the EVALUATE statement consists of a single literal, that operand is treated as a literal, not as an expression.
2. At the beginning of the execution of the EVALUATE statement, each selection subject is evaluated and assigned a value, a range of values, or a truth value as follows:
   A. Any selection subject specified by identifier-1 is assigned the value and class of the data item referenced by the identifier. If the selection subject is a numeric data item or a boolean data item whose length is one boolean position, the selection subject for this evaluation is treated as identifier-1 and not an arithmetic or boolean expression.
   B. Any selection subject specified by literal-1 is assigned the value and class of the specified literal.
   C. Any selection subject specified by arithmetic-expression-1 is assigned a numeric value according to the rules for evaluating an arithmetic expression.
   D. Any selection subject in which boolean-expression-1 is specified is assigned a boolean value according to the rules for evaluating boolean expressions.
   E. Any selection subject specified by condition-1 is assigned a truth value according to the rules for evaluating conditional expressions.
   F. Any selection subject 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.
3. The execution of the EVALUATE statement proceeds by processing each WHEN phrase from left to right in the following manner:
   A. Each selection object within the set of selection objects for each WHEN phrase is paired with the selection subject having the same ordinal position within the set of selection subjects. The result of the analysis of this set of selection subjects and objects is either true or false as follows:
      i. If the selection object is the word ANY, the result is true.
      ii. If the selection object is partial-expression-1, the selection subject is placed to the left of the leading relational operator and the resulting conditional expression is evaluated. The result of the evaluation is the truth value of the expression.
      iii. If the selection object is condition-2, the selection subject is either TRUE or FALSE. If the truth value of the selection subject and selection object match, the result of the analysis is true. If they do not match, the result is false.
      iv. If the selection object is either TRUE or FALSE, the selection subject is condition-1. If the truth value of the selection subject and selection object match, the result of the analysis is true. If they do not match, the result is false.
      v. If the selection object is a range-expression, the pair is considered to be a conditional expression of one of the following forms:

         when 'NOT' is not specified in the selection object;

            selection-subject >= left-part AND selection-subject <= right-part

when 'NOT' is specified in the selection object

selection-subject < left-part OR selection-subject > right-part

where left-part is identifier-3, literal-3, or arithmetic-expression-3 and right-part is identifier-4, literal-4, or arithmetic-expression-4. The result of the analysis is the truth value of the resulting conditional expression.

vi. If the selection object is identifier-2, literal-2, arithmetic-expression-2, or boolean-expression-2, the pair is considered to be a conditional expression of the following form:

selection-subject [NOT] = selection-object

where 'NOT' is present if it is present in the selection object. The result of the analysis is the truth value of the resulting conditional expression.

B. If the result of the analysis is true for every pair in a WHEN phrase, that WHEN phrase satisfies the set of selection subjects and no more WHEN phrases are analyzed.

C. If the result of the analysis is false for any pair in a WHEN phrase, no more pairs in that WHEN phrase are evaluated and the WHEN phrase does not match the set of selection subjects.

D. This procedure is repeated for subsequent WHEN phrases, in the order of their appearance in the source element, until either a WHEN phrase satisfying the set of selection subjects is selected or until all sets of selection objects are exhausted.

4. The execution of the EVALUATE statement then 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 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.

Examples

Evaluate logical expression, check if it is true or false

```
evaluate 1 = 1
when true  display message "Of course, it is True!"
when false display message "No way to be False!"
end-evaluate
```

Evaluate variable, check for different possible ranges of values

```
evaluate ws-age
when 1  thru 17 display message "Invalid Age for Credit Card Verification!"
when 18 thru 59 display message "Welcome to the Credit Card Verification App!"
when 60 thru 99 display message "Welcome to Senior Card Verification App!"
end-evaluate
```

Evaluate arithmetic expression, check for different possible values

```
evaluate ws-amount + 1
when 10 display "Ok"
when 11 display "A little bit high"
when 12 display "Value is high"
when 13 thru 99 display "Value is too high"
when other display "Invalid value"
end-evaluate
```

EXAMINE

Format 1

```
EXAMINE identifier-1 TALLYING {ALL        } literal-1 [REPLACING BY literal-2]
                              {LEADING    }
                              {UNTIL FIRST}
```

Format 2

```
EXAMINE identifier-1 REPLACING {ALL        } literal-1 BY literal-2
                               {FIRST      }
                               {LEADING    }
                               {UNTIL FIRST}
```

General Rules

1. The special register TALLY in Format 1 is described as PIC 9(5) COMP-N. TALLY is a count which represents a value that is dependant on the keywords following the word TALLYING.
2. If TALLYING UNTIL FIRST is specified, the integer in the TALLY register after execution of an EXAMINE statement is the number of occurrences of characters in identifier-1 before the first occurrence of literal-1.
3. If TALLYING ALL is specified, every occurrence of literal-1 is counted and the result of this counting is placed in the TALLY register.
4. If TALLYING LEADING is specified, only those occurrences of literal-1 that precede any other characters in the data item named by identifier are counted. For example, if the first character of identifier is not literal-1, the EXAMINE statement ceases execution immediately.
5. If the REPLACING phrase is used in conjunction with the TALLYING phrase, then, depending upon which keywords are used with the TALLYING phrase, those occurrences of literal-1 that participate in the tallying are replaced by literal-2.

REPLACING Phrase

6. The REPLACING phrase acts in the same manner as the REPLACING verb in the TALLYING phrase. However, since no tallying takes place, the TALLY register remains unchanged. The rules of the REPLACING phrase are stated below:
   A. If REPLACING ALL is specified, all occurrences of literal-1 in identifier-1 are replaced by literal-2.
   B. If REPLACING FIRST is specified, only the first occurrence of literal-1 is replaced by literal-2. If literal-1 does not appear in the data item represented by identifier-1, the data item is unchanged after execution of the EXAMINE statement.
   C. If REPLACING LEADING is specified, each occurrence of literal-1 is replaced by literal-2 until the first occurrence of a character other than literal-1 or the rightmost character of the data item is examined.

D. If REPLACING UNTIL FIRST is specified, every character of the data item represented by identifier-1 is replaced by literal-2 until literal-1 is encountered in the data item. If literal-1 does not appear in the data item, the entire data item is filled with literal-2.

Note - this statement is supported only with the -cv compiler flag.

Examples

Format 1 - Count how many "," characters are in the string

```
move "abc,cde,fgh,ijk" to ws-str
examine ws-str tallying all ","
*> Result in special register TALLY : 3
```

Format 1 - Count how many "," characters are in the string and replace them by "|"

```
move "abc,cde,fgh,ijk" to ws-str
examine ws-str tallying all "," replacing by "|"
*> Result in special register TALLY : 3, new value of ws-str : abc|cde|fgh|ijk
```

Format 2 - Replace all "," by "|" on a string

```
move "abc,cde,fgh,ijk" to ws-str
examine ws-str replacing all "," by "|"
*> Result in ws-str : abc|cde|fgh|ijk
```

Format 2 - Replace first "|" by "," on a string

```
move "abc|cde|fgh|ijk" to ws-str
examine ws-str replacing first "|" by ","
*> Result in ws-str : abc,cde|fgh|ijk
```

EXEC

General Format

```
EXEC SQL Sql-Statement
  END-EXEC
```

Where the sql statement are:

```
ALLOCATE
ALTER
BEGIN
CALL
CLOSE
COMMIT
CONNECT
CREATE
DECLARE
DELETE
DISCONNECT
DROP
EXECUTE
FETCH
FREE
GRANT
INCLUDE
INSERT
OPEN
PREPARE
REVOKE
ROLLBACK
SELECT
SET CONNECTION
UPDATE
WHENEVER
```

General Rules

1. ESQL support is built into the isCOBOL compiler. No separate pre-compiler is required to access a RDBMS using embedded SQL (i.e. EXEC SQL statements).

EXHIBIT

The EXHIBIT statement causes an (optionally conditional) display of the literals, and/or variables (optionally preceded by the variable name) specified in the statement.

General Format

```
EXHIBIT [NAMED] [CHANGED] {Literal-1    }  ...
                          {Identifier-1 }
```

Syntax rules

1. CHANGED is treated as commentary.

General rules

1. If NAMED is not used, each literal and variable value is displayed.
2. If NAMED is used, each literal is displayed, and each variable is displayed. Variables are preceded by "variable-name=" (where "variable-name" is replaced with the name of the variable in the EXHIBIT statement).

Note - this statement is supported only with the -cv compiler flag.

Examples

Format 1 - Display a variable and its value

```
       WORKING-STORAGE SECTION.
       77 my-var pic x(3) value "ABC".

       PROCEDURE DIVISION.

       MAIN.
           exhibit named my-var. |it will display: MY-VAR=ABC
           goback.
```

EXIT

Format 1

```
EXIT
```

Format 2

```
EXIT PROGRAM [ {RETURNING} {Literal-1   } ]
               {GIVING   } {Identifier-1}
```

Format 3

```
EXIT PERFORM [CYCLE]
```

Format 4

```
EXIT {PARAGRAPH}
     {SECTION  }
```

Format 5

```
EXIT METHOD
```

Syntax rules

Format 1

1. The EXIT statement shall appear in a sentence by itself that shall be the only sentence in the paragraph.

Format 2

2. Identifier-1 is a sending operand. It must me a numeric or data item.
3. Literal-1 is a sending operand. It must be a numeric literal.
4. GIVING and RETURNING are synonymous.

5. An EXIT PROGRAM statement may be specified only in a program procedure division.

Format 3

6. The EXIT PERFORM statement may be specified only in an inline PERFORM statement.

Format 4

7. The EXIT statement with the SECTION phrase may be specified only in a section.

Format 5

8. An EXIT METHOD statement may be specified only in a method procedure division.

General Rules

Format 1

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

2. If the EXIT PROGRAM statement is executed in a program that is not under the control of a calling runtime element, the EXIT PROGRAM statement is treated as if it were a CONTINUE statement.
3. The execution of an EXIT PROGRAM statement causes the executing program to terminate, and control to return to the calling statement. If a RETURNING or GIVING phrase is specified, its value it is assigned to return-code (runtime register) before the program is exited and it is returned to the invoking call if the RETURNING or GIVING clause is also defined on a CALL statement.

   If the calling runtime element is a COBOL element, execution continues in the calling element as specified in the rules for the CALL statement. The state of the calling runtime element 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 files shared between the calling runtime element 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 runtime element.

Format 3

4. The execution of an EXIT PERFORM statement without the CYCLE phrase causes control to be passed to an implicit CONTINUE statement immediately following the END-PERFORM phrase that matches the most closely preceding, and as yet unterminated, inline PERFORM statement.
5. The execution of an EXIT PERFORM statement with the CYCLE phrase causes control to be passed to an implicit CONTINUE statement immediately preceding the END-PERFORM phrase that matches the most closely preceding, and as yet unterminated, inline PERFORM statement.

Format 4

6. The execution of an EXIT PARAGRAPH statement causes control to be passed to an implicit CONTINUE statement immediately following the last explicit statement of the current paragraph, preceding any return mechanisms for that paragraph.

   NOTE - The return mechanisms mentioned in the rules for EXIT PARAGRAPH and EXIT SECTION are those associated with language elements such as PERFORM, SORT and USE.
7. The execution of an EXIT SECTION statement causes control to be passed to an unnamed empty paragraph immediately following the last paragraph of the current section, preceding any return mechanisms for that section.

Format 5

8. The execution of an EXIT METHOD statement causes the executing method to terminate, and control to return to the invoking statement. If a RETURNING phrase is present in the containing method definition, the value in the data item referenced by the RETURNING phrase becomes the result of the method invocation.

Examples

Format 1 - Exit placed at the end of a paragraphs block

```
procedure division.
main.
     perform elab-1 thru elab-1-exit.
     goback.
elab-1.
     add 1 to w-item.
elab-1-exit.
     exit.
```

Format 2 - Exit the program

```
exit program
```

Format 3 - Exit a perform loop

```
perform until 1 = 1 *> Would be infinite loop
  read customers next
       at end exit perform *> Loop is ended here
  end-read
  display cust-code cust-name
end-perform
```

Format 4 - Exit a paragraph

```
process-interest.
  if ws-amount not > 0
     exit paragraph
  end-if
  compute ws-interest = ws-amount * ws-rate / 100 / 12
  move ws-interest to loan-interest
  .
```

Format 5 - Exit a method at the first valid condition

```
method-id. check-age as "checkAge".
working-storage section.
77 age-range pic 9.
88 invalid-age value 0.
88 is-child    value 1.
88 is-teen     value 2.
88 is-adult    value 3.
linkage section.
77 age pic 9(3).
procedure division using age returning age-range.
main.
   if age = 0
      set invalid-age to true
      exit method
   end-if
   if age > 0 and age < 14
      set is-child to true
      exit method
   end-if
   if age > 13 and age < 20
      set is-teen to true
      exit method
   end-if
   if age > 19
      set is-adult to true
   end-if
   goback.
 end method.
```

GENERATE

General Format

```
GENERATE { data-name-1   }
         { report-name1 }
```

Syntax Rules

1. Data-name-1 shall name a detail report group. It may be qualified by a report-name.
2. Report-name-1 may be used only if the referenced report description entry contains a CONTROL clause.
3. If data-name-1 is defined in a containing program, the report description entry in which data-name-1 is specified and the file description entry associated with that report description entry shall contain a GLOBAL clause.
4. If report-name-1 is defined in a containing program, the file description entry associated with report-name-1 shall contain a GLOBAL clause.

General Rules

1. Execution of a GENERATE data-name statement causes one instance of the specified detail to be printed, following any necessary control break and page fit processing, as detailed below.
2. Execution of a GENERATE report-name-1 statement results in the same processing as for a GENERATE data-

name statement for the same report, except that no detail is printed. This process is called summary reporting. If all the GENERATE statements executed for a report between the execution of the INITIATE and TERMINATE statements are of this form, the report that is produced is called a summary report.

3. If a CONTROL clause is specified in the report description entry, each execution of the GENERATE statement causes the specified control data items to be saved and subsequently compared so as to sense for control breaks. This processing is described by the CONTROL clause.
4. Execution of the chronologically first GENERATE statement following an INITIATE causes the following actions to take place in order:
   A. The report heading is printed if defined. If the report heading appears on a page by itself, an advance is made to the next physical page, and PAGE-COUNTER is either incremented by 1 or, if the report heading's NEXT GROUP clause has the WITH RESET phrase, set to 1.
   B. A page heading is printed if defined.
   C. If a CONTROL clause is defined for the report, each control heading is printed, wherever defined, in order from major to minor.
   D. The specified detail is printed, unless summary reporting is specified.
5. Execution of any chronologically second and subsequent GENERATE statements following an INITIATE causes the following actions to take place in order:
   A. If a CONTROL clause is defined for the report, and a control break has been detected, each control footing and control heading is printed, if defined, up to the level of the control break.
   B. The specified detail is printed, unless summary reporting is specified.
6. If the associated report is divided into pages, the chronologically first body group since the INITIATE is preceded by a page heading, if defined, and the printing of any subsequent body groups is preceded by a page fit test which, if unsuccessful, results in a page advance, consisting of the following actions in this order:
   A. If a page footing is defined it is printed.
   B. An advance is made to the next physical page.
   C. If the associated report description entry contains a CODE clause with an identifier operand, the identifier is evaluated.
   D. If the page advance was preceded by the printing of a group whose description has a NEXT GROUP clause with the NEXT PAGE and WITH RESET phrases, PAGE-COUNTER is set to 1; otherwise PAGE-COUNTER is incremented by 1.
   E. LINE-COUNTER is set to zero.
   F. If a page heading is defined it is printed.
7. The report associated with data-name-1 or report-name-1 shall be in the active state. If it is not, the EC-REPORT-INACTIVE exception condition is set to exist.
8. If a non-fatal exception condition is raised during the execution of a GENERATE statement, execution resumes at the next report item, line, or report group, whichever follows in logical order.

Examples

A code snippet from the REPORT-SECTION.cbl sample program installed with isCOBOL under sample/data-access/files

```
...
        report section.
        rd  my-report
           controls are group-name
           page limit is 22
           heading 1
           first detail 4
           last detail 18
           footing 20.
...
        01  detail-line type is detail.
           02 line is plus 1.
              03 column 2    pic x(4) source prod-code.
              03 column 20   pic x(10) source prod-description.
              03 column 40   pic -$$,$$$.99 source prod-price.
...
        procedure division.
...
            generate detail-line.
...
```

GOBACK

General Format

```
GOBACK [ {RETURNING} Return-Value ]
         {GIVING   }
```

Syntax rules

1. `GIVING` and `RETURNING` are synonymous.
2. Identifier-1 must be a numeric literal or data item.

General Rules

1. If a GOBACK statement is executed in a program that is under the control of a calling runtime element, 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 runtime element, the program operates as if executing a STOP statement.
3. If a GOBACK statement is executed in a method, the method operates as if executing an EXIT METHOD statement.

Examples

Exit the program with exit status 1

```
goback 1
```

GO TO

Format 1

```
GO TO Procedure-Name-1
```

Format 2

```
GO TO { Procedure-Name-1 } ... DEPENDING ON Identifier-1
```

Syntax rules

1. Identifier-1 shall reference a numeric elementary data item that is an integer.

General rules

1. When a GO TO statement represented by format 1 is executed, control is transferred to procedure-name-1.
2. When a GO TO statement represented by format 2 is executed, control is transferred to procedure-name-1, etc., depending on the value of identifier-1 being 1, 2, ... , n.

Examples

Format 1 - Go to a paragraph if a condition is true

```
compute-int section.
validate.
  if ws-amount > 0 go to compute-interest
  else go to exit-compute.

compute-interest.
  compute ws-interest = ws-amount * ws-rate / 100 / 12
  move ws-interest to loan-interest
  .

exit-compute.
  exit.
```

Format 2 - Go To a paragraph depending on menu option selected

```
process-menu section.
accept-option.
  accept ws-option
  go to op1 op2 op3 depending on ws-option.

op1.
  display message "processing op1"
  go to process-exit.

op2.
  display message "processing op2"
  go to process-exit.

op3.
  display message "processing op3"
  go to process-exit.

process-exit.
  exit.
```

IF

General Format

```
IF Condition THEN { statement-1   }
                  { NEXT SENTENCE }

  [ ELSE statement-2 [END-IF] ]
  [ ELSE NEXT SENTENCE        ]
  [ END-IF                    ]
```

Syntax rules

1. Statement-1 and statement-2 represent either one or more imperative statements or a conditional statement optionally preceded by one or more imperative statements.
2. Statement-1 and statement-2 may each contain an IF statement. In this case, the IF statement is said to be nested.

   IF statements within IF statements are considered matched IF, ELSE, and END-IF ordered combinations, proceeding from left to right. 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 preceding IF that has not been implicitly or explicitly terminated.
3. The ELSE NEXT SENTENCE phrase may be omitted if it immediately precedes the terminal separator period of the sentence.

General rules

1. If Condition is true, control is transferred to the first statement of statement-1 and execution continues according to the rules for each statement specified in statement-1. The ELSE phrase, if specified, is ignored.
2. If Condition is false, the THEN phrase is ignored. If the ELSE phrase is specified, control is transferred to the first statement of statement-2 and execution continues according to the rules for each statement specified in

statement-2.

3. If 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. The ELSE phrase, if specified, is ignored.
4. If Condition is true and NEXT SENTENCE is specified in the THEN phrase, the ELSE phrase, if specified, is ignored and control is transferred to an implicit CONTINUE statement immediately preceding the next separator period.
5. If Condition is false and statement-2 is specified, the THEN phrase 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.
6. If Condition is false and NEXT SENTENCE is specified in the ELSE phrase, the THEN phrase is ignored and control is transferred to an implicit CONTINUE statement immediately preceding the next separator period.
7. If Condition is false and the ELSE phrase is not specified, the THEN phrase is ignored.

Examples

Evaluate condition to process interest computation

```
if ws-amount not > 0
   exit paragraph
else
   perform compute-interest
end-if
display message "Interest computed : " ws-interest
```

INITIALIZE

Format 1

```
INITIALIZE { Identifier-1 } ... [ WITH FILLER ]

  [ REPLACING { {ALPHABETIC         } DATA BY Identifier-2 } ... ]
              {ALPHANUMERIC       }
              {NUMERIC            }
              {ALPHANUMERIC-EDITED}
              {NUMERIC-EDITED     }
  [ {ALL           } TO VALUE ]
    {category-names}
  [ THEN REPLACING { ALPHABETIC          } DATA BY { identifier-3 } ]
                   { ALPHANUMERIC        }
                   { ALPHANUMERIC-EDITED }
                   { BOOLEAN             }
                   { NATIONAL            }
                   { NATIONAL-EDITED     }
                   { NUMERIC             }
                   { NUMERIC-EDITED      }
                   { OBJECT-REFERENCE    }
  [ THEN TO DEFAULT ]
```

Format 2

```
INITIALIZE  Identifier-4 [ WITH SIZE Identifier-5 ]
```

Syntax Rules

Format 1

1. If the REPLACING phrase is used, a MOVE statement with identifier-2 or literal-1 as the sending item and an item of the specified category as the receiving operand shall be valid.
2. The data description entry for the data item referenced by identifier-1 shall not contain a RENAMES clause.
3. The data item referenced by identifier-1 is the receiving operand.
4. If the REPLACING phrase is specified, literal-1 or the data item referenced by identifier-2 is the sending operand. If the REPLACING phrase is not specified, the sending operand is determined according to the general rules of the INITIALIZE statement.
5. The same category shall not be repeated in a REPLACING phrase.
6. The keywords in category-names correspond to a category of data: alphabetic, alphanumeric, alphanumeric-edit, numeric, numeric-edited. If ALL is specified in the VALUE phrase it is as if all of the categories listed in category-names were specified.

Format 2

7. Identifier-4 must be an ANY LENGTH item.
8. Identifier-5 can be any numeric data item or literal.

General Rules

Format 1

1. When identifier-1 references a bit group item or a national group item, identifier-1 is processed as a group item. When identifier-2 references a bit group item or a national group item, identifier-2 is processed as an elementary data item.
2. If more than one identifier-1 is specified in an INITIALIZE statement, the result of executing this INITIALIZE statement is the same as if a separate INITIALIZE statement had been written for each identifier-1 in the same order as specified in the INITIALIZE statement.
3. Whether identifier-1 references an elementary item or a group item, the effect of the execution of the INITIALIZE statement is as though a series of implicit MOVE or SET statements, each of which has an elementary data item as its receiving operand, were executed.

   The implicit statement is:

   MOVE sending-operand TO receiving-operand.
4. The receiving-operand in each implicit MOVE or SET statement is determined by applying the following steps in order:
   A. First, the following data items are excluded as receiving-operands:
      i. Any identifiers that are not valid receiving operands of a MOVE statement.
      ii. If the FILLER phrase is not specified, elementary data items with an explicit or implicit FILLER clause.
      iii. Any elementary data item subordinate to identifier-1 whose data description entry contains a REDEFINES or RENAMES clause or is subordinate to a data item whose data description entry contains a REDEFINES clause. However, identifier-1 may itself have a REDEFINES clause or be subordinate to a data item with a REDEFINES clause.
   B. Second, an elementary data item is a possible receiving item if:
      i. It is explicitly referenced by identifier-1; or
      ii. It is contained within the group data item referenced by identifier-1. If the elementary data item is a table element, each occurrence of the elementary data item is a possible receiving-operand.
   C. Finally, each possible receiving-operand is a receiving-operand if at least one of the following is true:
      i. The VALUE phrase is specified, the category of the elementary data item is one of the categories specified or implied in the VALUE phrase, and the following is true:
         a. A data-item format or table format VALUE clause is specified in the data description entry of the elementary data item.
      ii. The REPLACING phrase is specified and the category of the elementary data item is one of the categories specified in the REPLACING phrase.
      iii. The DEFAULT phrase is specified.
      iv. Neither the REPLACING phrase nor the VALUE phrase is specified.
5. The sending-operand in each implicit MOVE and SET statement is determined as follows:
   A. If the data item qualifies as a receiving-operand because of the VALUE phrase:
      i. The sending-operand is determined by the literal in the VALUE clause specified in the data description entry of the data item. If the data item is a table element, the literal in the VALUE clause that corresponds to the occurrence being initialized determines the sending-operand. The actual sending-operand is a literal that, when moved to the receiving-operand with a MOVE statement, produces the same result as the initial value of the data item as produced by the application of the VALUE clause.
   B. If the data item does not qualify as a receiving-operand because of the VALUE phrase, but does qualify because of the REPLACING phrase, the sending-operand is the literal-1 or identifier-2 associated with the category specified in the REPLACING phrase.

C. The sending-operand used depends on the category of the receiving-operand as follows:

Receiving operand	Figurative constant
Alphabetic	Figurative constant alphanumeric SPACES
Alphanumeric	Figurative constant alphanumeric SPACES
Alphanumeric-edited	Figurative constant alphanumeric SPACES
Boolean	Figurative constant ZEROES
National	Figurative constant national SPACES
National-edited	Figurative constant national SPACES
Numeric	Figurative constant ZEROES
Numeric-edited	Figurative constant ZEROES
Object-reference	Predefined object reference NULL

6. If identifier-1 references a group data item, affected elementary data items are initialized in the sequence of their definition within the group data item. For a variable-occurrence data item, the number of occurrences initialized is determined by the rules of the OCCURS clause for a receiving data item.
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.
8. Index data items and elementary FILLER data items are not affected by the INITIALIZE statement, unless the optional WITH FILLER phrase is specified, in which case FILLER data items are initialized.

Format 2

9. A Format 2 INITIALIZE statement clears and resize ANY LENGTH items.
10. Identifier-5 specifies the new size of Identifier-4; if the SIZE clause is omitted or if a negative size is specified, then a size of zero is assumed. After the resizing, all the characters in the item are set to space.

Examples

The following examples are applied to the following group data item

```
01 var1.
   05 num1   pic 9(3) value 125.
   05 str1   pic x(3) value "abc".
   05 num-ed pic zz,zz9 value 1234.
```

Initialize group item to the default values by type

```
*> values before initialize: 125abc 1,234
initialize var1
*> values after  initialize: 000        0
```

Initialize group item to specific values by type

```
*> values before initialize: 125abc 1,234
initialize var1 replacing
                alphanumeric   by all "x"
                numeric        by all "9"
                numeric-edited by "0"
end-initialize
*> values after  initialize: 999xxx     0
```

INITIATE

General Format

```
INITIATE report-name-1
```

Syntax Rules

1. Report-name-1 shall be defined by a report description entry in the report section.
2. If report-name-1 is defined in a containing program, the file description entry associated with report-name-1 shall contain a GLOBAL clause.

General Rules

1. The INITIATE statement initializes
    - all the report's sum counters to zero
    - the report LINE-COUNTER to zero
    - the report PAGE-COUNTER to one

Before the INITIATE statement is executed the file associated with the Report must have been opened for OUTPUT or EXTEND.

Examples

A code snippet from the REPORT-SECTION.cbl sample program installed with isCOBOL under sample/data-access/files

```
...
       report section.
       rd  my-report
           controls are group-name
           page limit is 22
           heading 1
           first detail 4
           last detail 18
           footing 20.
...
       PROCEDURE DIVISION.
...
           initiate my-report.
...
```

INQUIRE

Format 1

```
INQUIRE {Control-Item       } [ ( {Index-1} ... ) ]
        {CONTROL AT Location}

  { { Property-Name          } [IN] { [MULTIPLE] Property-Value [ LENGTH {IN} Length-
1 ] } } ...
    { PROPERTY Property-Type }      { [TABLE   ]                          {= }            }

  { CLASS [IN] Control-Class  }
  { HANDLE [IN] Control-Handle}
  { STATUS [IN] Control-Status}
```

Location is defined as follows:

```
{ Screen-Loc [CELL  ]
             [CELLS ]
             [PIXEL ]
             [PIXELS]


  LINE NUMBER Line-Num [CELL  ]
                       [CELLS ]
                       [PIXEL ]
                       [PIXELS]

  {COLUMN  } NUMBER Col-Num [CELL  ]
  {COL     }                [CELLS ]
  {POSITION}                [PIXEL ]
  {POS     }                [PIXELS]

  }
```

Format 2

```
INQUIRE { window-handle           } { Property-Name          [IN] Property-Value }
        { WINDOW [generic-handle] } { STATUS                 [IN] Control-Status }
```

Syntax rules

1. CELL and CELLS, PIXEL and PIXELS and . COLUMN, COL, POSITION and POS are synonymous.
2. Control-item is defined as USAGE HANDLE and identifies the control being inquired.
3. Index-1 is numeric and defined as numeric, with parenthesis as when referencing a table index.
4. The AT, LINE, COLUMN, CLINE, and CCOL phrases must be used if the CONTROL phrase is specified.
5. Screen-loc is an integer data item or literal.
6. Line-num, col-num, are numeric data items or literals.
7. Property-name is the name of a property specific to the type of control being inquired. Refer to Graphical Control List for the list of properties supported by a specific control type. If control-item refers to a generic handle, or if the CONTROL option is specified, then property-name cannot be used. Use the PROPERTY phrase instead.

8. Property-type is a numeric literal or data item. Possible values are listed in the iscontrols.def copybook.
9. Property-value is a data item.
10. Length-1 is a numeric data item. The LENGTH phrase may be specified only if the value or property-value immediately preceding it is an alphanumeric data item.
11. Control-Class is an alphanumeric data item.
12. Control-Handle is a USAGE HANDLE data item.
13. Control-Status is a numeric data item.

General rules

1. The INQUIRE statement retrieves some or all of a control's current properties, and stores them as data items. *Control-item* identifies the control to inquire. If the CONTROL phrase is used instead, the runtime inquires the control located at the screen position specified by the AT, LINE, and COLUMN phrases in the current window. A list of controls is maintained for each window. This list is kept in the order that controls are created.
2. If *control-item* does not refer to a valid control, or if the runtime cannot locate a control at the specified screen location, the INQUIRE statement returns the "Invalid Handle" error or has no effect if `iscobol.ignore_invalid_handle` is set to true.
3. If index-1 is specified, the properties described below are modified to match the value of index-1. This happens before the inquiry executes. The Control properties updated are shown below:

Control Type	Properties Affected
List Box	QUERY-INDEX
Grid	Y, X
Tree View	ITEM

   Each occurrence of index-1 modifies one property.

4. The LENGTH option, length-1 returns the exact number of characters placed by the control in value or property-value.
5. The LINE NUMBER IN and COLUMN NUMBER IN phrases return the location of the control in line-num and col-num respectively. The SIZE IN and LINES IN phrases return the dimensions of the control in width and height respectively. Use LINE and SIZE dimension values twhen creating the CONTROL in order to have their updated values returned with INQUIRY.
6. The CLASS clause returns the control type. Possible values, defined in iscontrols.def, are

Value	Control type
1	Label
2	Entry-Field
3	Push-Button
4	Check-Box
5	Radio-Button
6	Scroll-Bar

Value	Control type
7	List-Box
8	Combo-Box
9	Frame
10	Tab-Control
11	Bar
12	Grid
13	Bitmap
14	Tree-View
15	Web-Browser
17	Status-Bar
18	Date-Entry
21	Slider
22	Java-Bean
23	Ribbon
24	Scroll-Pane

7. The HANDLE clause returns the control handle.
8. The STATUS clause returns a numeric value that describes the status of the control or the window. Possible values are:

STATUS	Meaning
0	the handle is valid, the control or window is currently on video
1	the handle is not valid
2	the control or window isn't yet created or has been destroyed by destroy verb
3	the control or window has been destroyed internally by the framework (like it happens with temporary controls, for example) and not by a destroy verb

Examples

Format 1 - Get the value from a entry-field to a variable

```
inquire scr-cust-code value ws-cust-code
```

Format 1 - Get the class that a generic handle points to; if it's an entry-field, then get the max-text value.

```
working-storage section.
copy "iscontrols.def".
77 generic-handle handle.
77 ctl-class pic 9(5).
77 wrk-mt    pic 9(5).
...
procedure division.
...
inquire generic-handle class in ctl-class.
if ctl-class = ctl-entry-field
   inquire generic-handle property efp-max-text in ws-max-text
end-if.
```

Format 2 - Get the size in columns and lines of a graphical window

```
inquire mywin-handle size ws-win-size
                     lines ws-win-lines
```

INSPECT

Format 1

```
INSPECT identifier-1 TALLYING tallying-phrase
```

Format 2

```
INSPECT identifier-1 REPLACING replacing-phrase
```

Format 3

```
INSPECT identifier-1 TALLYING tallying-phrase REPLACING replacing-phrase
```

Format 4

```
INSPECT identifier-1 CONVERTING {identifier-6} TO {identifier-7} [after-before-phrase]
                                {literal-4   }    {literal-5   }
```

where tallying-phrase is:

```
{ identifier-2 FOR {CHARACTERS [ after-before-phrase ]           } ... } ...
                   {ALL { {identifier-3} [ after-before-phrase ] } ... }
                        {literal-1   }
                   {LEADING { {identifier-3} [ after-before-phrase ] ... }
                            {literal-1   }
                   {TRAILING { {identifier-3} [ after-before-phrase ] ... }
                             {literal-1   }
```

where after-before-phrase is:

```
{ [ AFTER INITIAL  {identifier-4} ]            }
                   {literal-2   }
{ [ BEFORE INITIAL [TRAILING] {identifier-4} ] }
                              {literal-2   }
```

where replacement-phrase is:

```
{ CHARACTERS                BY replacement-item [ after-before-phrase ]       } ...
{ ALL      { {identifier-3} BY replacement-item [ after-before-phrase ] } ... }
             {literal-1   }
{ LEADING  { {identifier-3} BY replacement-item [ after-before-phrase ] } ... }
             {literal-1   }
{ FIRST    { {identifier-3} BY replacement-item [ after-before-phrase ] } ... }
             {literal-1   }
{ TRAILING { {identifier-3} BY replacement-item [ after-before-phrase ] } ... }
             {literal-1   }
```

where replacement-item is:

```
{ identifier-5 }
{ literal-3    }
```

Syntax Rules

All Formats

1. Identifier-1 shall reference either an alphanumeric or national 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, boolean, 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 is national; when identifier-1 is of class boolean, the figurative constant is of class boolean and only the figurative constant ZERO may be specified; otherwise, the class of the figurative constant is 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 boolean or national, then all shall reference a data item or literal of class boolean or national, respectively.

Formats 1 AND 3

5. Identifier-2 shall reference an elementary numeric data item.
6. the FOR clause can be repeated in the tallying-phrase to specify different criteria.

Formats 2 AND 3

7. 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.
8. When the CHARACTERS phrase is specified, literal-3 shall be one character in length.

Format 1

9. Identifier-1 is a sending operand.

Format 4

10. 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. If the data item referenced by identifier-1 is a zero-length item, execution of the INSPECT statement terminates immediately.
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 an alphanumeric-edited data item, or a numeric-edited or unsigned numeric data item described explicitly or implicitly with usage display, 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 national-edited data item, or a numeric-edited or unsigned numeric data item described explicitly or implicitly with usage national, the data item is 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 or 3c 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

6. 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).
7. 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:

   i. If neither LEADING nor FIRST is specified; or

   ii. If the LEADING adjective applies to literal-1 and literal-1 is a leading occurrence as defined in general rules 11 and 16; or

   iii. 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.

8. 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 or identifier-4 references a zero-length item, 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. If the TRAILING clause is specified, all the rightmost contiguous characters identified by Literal-2 will not be included. 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

9. Both the ALL and LEADING phrases are transitive across the operands that follow them until another All or LEADING phrase is encountered.
10. The content of the data item referenced by identifier-2 is not initialized by the execution of the INSPECT statement.
11. 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. The TRAILING phrase is used to find the rightmost occurrence, or set of contiguous occurrences, in identifier-1. If a TRAILING occurrence is found, a right to left scan of identifier-1 is made to find contiguous occurrences.
12. 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.
13. If multiple FOR clauses are specified in the tallyling-phrase, the runtime resolves each one of them by analyzing Identifier-1 from the position of the previous verified condition. If -ca compiler option is used, instead, the runtime analyzes Identifier-1 from the beginning when resolving each one of the FOR clauses.

Formats 2 and 3

14. 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.
15. When the CHARACTERS phrase is used, the data item referenced by identifier-5 shall be one character in length.

Format 2

16. The ALL, FIRST, and LEADING phrases are transitive across the operands that follow them until another All, FIRST, or LEADING phrase is encountered.
17. 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. If the TRAILING phrase is present, the REPLACING option causes all contiguous occurrences of identifier-3 to be replaced by replacement-item, provided that these occurrences end in the rightmost character position of identifier-1.

18. 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.

Format 3

19. 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. Item identification of any identifier in the format 2 statement is done only once before executing the format 1 statement.

Format 4

20. 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.
21. 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.
22. 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. If these sizes are not equal, the results of the execution of the INSPECT statement are undefined.
23. 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.

Examples

Format 1 - Counting some capital letters on a string

```
initialize ws-count
move    "Another Beautiful Day" to ws-str
inspect ws-str tallying ws-count
        for all "A" "B" "C" "D" "E" "F"
*> Value of ws-count : 3
```

Format 2 - Changing "a" by "o" not from the beginning but after another character.

```
move "a first sentence with a. Hella Warld!" to ws-str
inspect ws-str replacing all "a" by "o" after initial "."
*> New value for ws-str: a first sentence with a. Hello World!
```

Format 2 - Replace all characters in string by zeroes

```
77 ws-str pic x(15) value "hello world!".
...
inspect ws-str replacing characters by zero
*> New value for ws-str : 000000000000000
```

Format 2 - Replace characters by zeroes before the first quote

```
77 ws-str   pic x(30) value 'hello world! "do not change"'.
77 ws-str   pic x(30) value 'hello world! "do not change"'.
...
inspect ws-str replacing
        characters by zeros
        before initial quote
*> New value for ws-str : 0000000000000"do not change"
```

Format 3 - Count the leading zeroes on a string and also change the first "a" by "2" after a specific character

```
initialize ws-count
move "00academy00" to ws-str
inspect ws-str tallying ws-count for leading "0"
        replacing first "a" by "2"
        after initial "c"
*> New value for ws-count : 2
*> New value for ws-str   : 00ac2demy00
```

Format 4 - Converting Uppercase to Lowercase

```
move "THIS IS THE SENTENCE" to ws-str
inspect ws-str
        converting "ABCDEFGHIJKLMNOPQRSTUVWXYZ" to
                   "abcdefghijklmnopqrstuvwxyz"
*> New value for ws-str : this is the sentence
```

INVOKE

General Format

```
INVOKE {Identifier-1} Method-name-1
       {Class-name-1}
       {SELF        }
       {SUPER       }

  [ USING { [ BY {REFERENCE} ] { {Identifier-3 } [ AS {Class-Name-1} ] } ... } ... ]
                 {CONTENT  }     {Literal-2    }      {native-type }
                 {VALUE    }     {Arith-Expr-1 }
                                 {OMITTED      }
                                 {NULL         }
                                 {TRUE         }
                                 {FALSE        }

  [ {RETURNING} INTO Identifier-4 ] [ AS Class-Name-1 ]
    {GIVING   }

  [ ON {EXCEPTION} Imperative-Statement-1]
       {OVERFLOW }

  [ NOT ON {EXCEPTION} Imperative-Statement-2]
           {OVERFLOW }

[END-INVOKE]
```

Alternative syntaxes

The INVOKE statement can be expressed with the following alternate syntax:

```
INVOKE    {Identifier-1}{:>} Method-name-1
          {Class-name-1}{::}
          {SELF        }
          {SUPER       }

  [ ( { [ BY {REFERENCE} ] { {Identifier-3 } [ AS {Class-Name-1} ] } ... } ...) ]
             {CONTENT  }     {Literal-2    }      {native-type }
             {VALUE    }     {Arith-Expr-1 }
                             {OMITTED      }
                             {NULL         }
                             {TRUE         }
                             {FALSE        }

  [ {RETURNING} INTO Identifier-4 ] [ AS Class-Name-1 ]
    {GIVING   }

  [ ON {EXCEPTION} Imperative-Statement-1]
       {OVERFLOW }

  [ NOT ON {EXCEPTION} Imperative-Statement-2]
           {OVERFLOW }

[END-INVOKE]
```

The same feature provided by the INVOKE statement can be obtained with the following alternate syntax:

```
[TRY]
  [ SET Identifier-4 [ AS Class-Name-1 ] {TO} ]
                                         {=}
   {Identifier-1}{:>} Method-name-1
   {Class-name-1}{::}
   {SELF         }
   {SUPER        }
          [({ [ BY {REFERENCE} ] { {Identifier-3 } [ AS {Class-Name-1} ] } ... } ...)]
                   {CONTENT  }     {Literal-2    }      {native-type }
                   {VALUE    }     {Arith-Expr-1 }
                                   {OMITTED      }
                                   {NULL         }
                                   {TRUE         }
                                   {FALSE        }
   Imperative-Statement-2
[CATCH EXCEPTION Imperative-Statement-1]
[END-TRY]
```

Syntax rules

1. Identifier-1 shall be an object reference.
2. Method-name-1 shall be a alphanumeric or national data item or literal. When the separator between Identifier-1 and Method-name-1 is not a space, the literal can be expressed without quotes.
3. If Class-name-1 is specified, literal-1 shall be specified. The value of Method-name-1 shall be the name of a method defined in the factory interface of Class-name-1.
4. native-type can be any of the following keywords: BOOLEAN, BYTE, CHAR, DOUBLE, FLOAT, INT, LONG, SHORT, STRING
5. If identifier-1 is specified and it does not reference a universal object reference, Method-name-1 shall be specified. The value of Method-name-1 shall be the name of a method, subject to the following conditions:
   A. If identifier-1 references an object reference described with a class-name and the FACTORY phrase, Method-name-1 shall be the name of a method contained in the factory interface of that class-name.
   B. If identifier-1 references an object reference described with a class-name without the FACTORY phrase, Method-Name-1 shall be the name of a method contained in the instance interface of that class-name.
   C. If SELF and the method containing the INVOKE statement is a factory method, Method-name-1 shall be the name of a method contained in the factory interface of the class containing the INVOKE statement.
   D. If SELF and the method containing the INVOKE statement is an instance method, Method-name-1 shall be the name of a method contained in the instance interface of the class containing the INVOKE statement.
   E. If SUPER and the method containing the INVOKE statement is a factory method, Method-name-1 shall be the name of a method contained in the factory interface of a class inherited by the class containing the INVOKE statement.
   F. If SUPER and the method containing the INVOKE statement is an instance method, Method-name-1 shall be the name of a method contained in the instance interface of a class inherited by the class containing the INVOKE statement.
6. Identifier-3 shall be an address-identifier or shall reference a data item defined in the file, working-storage, or linkage section.
7. Identifier-3 shall not reference a data item defined in the file or working-storage section of a factory or instance object.

8. Identifier-4 shall reference a data item defined in the file, working-storage, or linkage section.
9. If identifier-3 references an address-identifier, identifier-3 is a sending operand.
10. If identifier-3 does not reference an address-identifier, identifier-3 is a receiving operand.
11. Arith-Expr-1 shall reference an arithmetic expression.
12. Identifier-4 is a receiving operand.
13. `GIVING` and `RETURNING` are synonymous.
14. `EXCEPTION` and `OVERFLOW` are synonymous.
15. `BY REFERENCE`, `BY VALUE` and `BY CONTENT` are supported only for standard COBOL variables and not for OBJECT REFERENCE items.
16. ":>" and "::" are synonymous.

General rules

1. The instance of the program, function, or method that executes the INVOKE statement is the activating runtime element.
2. Identifier-1 identifies an instance object. If class-name-1 is specified, it identifies the factory object of the class referenced by that class-name. Method-name-1 identifies a method of that object that will act upon that instance object.
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. This 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.

4. If identifier-1 is null, the execution of the INVOKE statement is terminated.
5. Identifier-3 must be the same type as the receiving operand in the method constructor.
6. Execution of the INVOKE statement proceeds as follows:
   A. Identifier-1 and identifier-3 are evaluated and item identification is done for identifier-4 at the beginning of the execution of the INVOKE statement. If an exception condition exists, no method is invoked and execution proceeds as specified in general rule 5E. If an exception condition does not exist, the values of identifier-3 or literal-2 are made available to the invoked method at the time control is transferred to that method.
   B. The runtime system attempts to locate the method being invoked. If the method is not found or the resources necessary to execute the method are not available, the method is not activated, and execution continues as specified in general rule 5E.
   C. The method specified by the INVOKE statement is made available for execution and control is transferred to the invoked method. Control is transferred to the invoked method in a manner consistent with the entry convention specified for the method.
   D. After control is returned from the invoked method, if an exception condition is propagated from the invoked method, execution continues as specified in general rule 5E.
   E. If an exception condition has been raised, if the ON EXCEPTION clause is defined, the control is transferred to imperative-statement-1; if the ON EXCEPTION clause is not defined, the exception will continue to propagate. If no exception condition has been raised, and if the NOT ON EXCEPTION clause is defined the control is transferred to imperative-statement-2 otherwise control is transfered to the end of the INVOKE
7. If a RETURNING phrase is specified, the result of the activated method is placed into identifier-4.

8. If an OMITTED phrase is specified, no argument is passed to the invoked method.
9. The AS clause allows you to perform a cast. When the data item is casted as STRING, its value is trimmed.
10. If a NULL phrase is specified, a null generic object reference is passed as argument to the invoked method.

Examples

Invoke methods of a Java Class to get System Information using different syntaxes:

```
configuration section.
repository.
    class Sys as "java.lang.System".

working-storage section.
77  buffer      pic x(50).

procedure division.
main.
    invoke Sys "getProperty" using "os.name" giving buffer.
    display "Operating System: " buffer.

    invoke Sys::"getProperty" ( "os.arch" ) giving buffer.
    display "OS Architecture: " buffer.

    invoke Sys::getProperty ( "java.version" ) returning buffer.
    display "Java Version: " buffer.

    set buffer to Sys:>getProperty ( "java.io.tmpdir" ).
    display "Java Version: " buffer.
```

JSON GENERATE

General Format

```
JSON GENERATE Identifier-1 FROM Identifier-2

     [ COUNT IN Identifier-3 ]

     [ NAME OF Identifier-4 IS Literal-1 ]

     [ SUPPRESS Identifier-5 ]

     [ ON EXCEPTION Imperative-Statement-1 ]

     [ NOT ON EXCEPTION Imperative-Statement-2 ]

 [ END-JSON ]
```

Syntax Rules

1. Identifier-1 must be either: an elementary data item of category alphanumeric; an alphanumeric group item; an elementary data item of category national; or a national group item.
2. Identifier-1 must not be defined with the JUSTIFIED clause.
3. Identifier-1 can be subscripted or reference modified. Identifier-4 or identifier-5 cannot be reference modified or subscripted.

4. Identifier-1 must be large enough to contain the generated JSON text. Typically, it should be from 2 to 3 times the size of identifier-2, depending on the lengths of the data-names within identifier-2.
5. If the SUPPRESS phrase is in effect, identifier-1 must still be large enough to contain the generated JSON text before suppression.
6. Identifier-3 is a numeric data item.
7. Literal-1 must be an alphanumeric or national literal containing the name to be generated in the JSON text corresponding to identifier-4.
8. Identifier-5 must reference a data item that is subordinate to identifier-2 and that is not otherwise ignored by the operation of the JSON GENERATE statement.

General Rules

1. Identifier-1 is the receiving area for the generated JSON text.
2. If identifier-1 is not large enough, an exception condition exists at the end of the JSON GENERATE statement. If the COUNT phrase is specified, identifier-3 contains the number of character encoding units that were actually generated.
3. Identifier-2 is the group or elementary data item to be converted to JSON format.
4. If the COUNT phrase is specified, Identifier-3 contains the count of generated JSON character encoding units. If Identifier-1 is of category national, the count is in double-bytes. Otherwise, the count is in bytes.
5. The NAME phrase enables you to override the default JSON names derived from Identifier-2 or its subordinate data items.
6. The SUPPRESS phrase enables you to selectively generate output for the JSON GENERATE statement by allowing you to identify and unconditionally exclude items that are subordinate to Identifier-2.
7. If identifier-5 specifies a group data item, that group data item and all subordinate data items are excluded. Duplicate specifications of identifier-5 are permitted.
8. If the ON EXCEPTION phrase is specified, control transfers to Imperative-statement-1 when an error occurs during generation of the JSON document.
9. If the ON EXCEPTION phrase is not specified, the NOT ON EXCEPTION phrase, if any, is ignored, and control is transferred to the end of the JSON GENERATE statement in the event of an error.
10. If an exception condition does not occur during generation of the JSON document, control is passed to Imperative-statement-2, if specified; otherwise, control is passed to the end of the JSON GENERATE statement.

Examples

Generate JSON code from group data item

```
       WORKING-STORAGE SECTION.
       01 GRP.
          05 AC-NO PIC AA9999 VALUE 'SX1234'.
          05 MORE.
             10 STUFF PIC S99V9 OCCURS 2.
          05 SSN PIC 999/99/9999 VALUE '987-65-4321'.
       01 RESULT PIC X(80).
       01 I BINARY PIC 99.
       PROCEDURE DIVISION.
       MAIN.
           MOVE 7.8 TO STUFF(1), MOVE -9 TO STUFF(2)
           JSON GENERATE RESULT FROM GRP COUNT I
             NAME OF STUFF IS 'Value' SUPPRESS SSN

           DISPLAY I
           DISPLAY RESULT(1:I)
           GOBACK.
***********
********** The example produces the following output:
********** 54
********** {"GRP":{"Ac-No":"SX1234","More":{"Value":[7.8,-9.0]}}}
```

JSON PARSE

General Format

```
JSON PARSE Identifier-1 INTO Identifier-2

     [ WITH DETAIL ]

     [ NAME OF Identifier-3 IS Literal-1 ]

     [ SUPPRESS Identifier-4 ]

     [ ON EXCEPTION Imperative-Statement-1 ]

     [ NOT ON EXCEPTION Imperative-Statement-2 ]

 [ END-JSON ]
```

Syntax Rules

1. Identifier-1 must reference one of the following: an elementary data item of category alphanumeric; an alphanumeric group item.
2. Identifier-1 must not be defined with the JUSTIFIED clause.
3. Identifier-1 can be subscripted or reference modified. Identifier-2 cannot be reference modified, but it can be subscripted.
4. Identifier-2 must be an alphanumeric group item, or elementary data item of category alphanumeric.
5. Identifier-3 must reference Identifier-2 or one of its subordinates.

6. Identifier-3 may be specified more than once, but only the last specification is used.
7. Literal-1 must be alphanumeric or a national literal containing the JSON name to be associated with identifier-3.
8. Identifier-4 must reference an item that is subordinate to Identifier-2.
9. Duplicate specifications of Identifier-4 are permitted.

General Rules

1. Identifier-1 is the data item that contains the JSON text.

1. If identifier-1 contains invalid JSON syntax, the statement terminates with an exception condition, and Identifier-2 may be partially modified.
2. Identifier-2 is the group or elementary item populated with the JSON text.
3. The NAME OF phrase enables you to use a different data name than the one required by the matching algorithm when matching the name of a JSON name/value pair to a COBOL data item. The NAME OF phrase must not result in an ambiguous name specification, and is in effect during the execution of the JSON PARSE statement.
4. The SUPPRESS phrase enables you to exclude items that are subordinate to Identifier-2 from being parsed.
5. When an exception occurs, if the ON EXCEPTION phrase is specified, control is passed to Imperative-statement-1, and the NOT ON EXCEPTION phrase, if specified, is ignored. If the ON EXCEPTION phrase is not specified, control is transferred to the end of the statement. Such conditions can occur when the JSON text is ill-formed or there are problems populating Identifier-2 with the JSON text.
6. If an exception does not occur, and the NOT ON EXCEPTION phrase is specified, control is passed to imperative-statement-2; otherwise control is transferred to the end of the statement, and special register JSON-CODE contains zero after statement execution.

Examples

Parse a JSON string and move the values to a group data item

```
       WORKING-STORAGE SECTION.
       01 JSON-DATA PIC X(80) VALUE
           '{"GRP":{"Ac-No":"SX1234","More":{"Value":[7.8,-9.0]}}}'.
       01 GRP.
          05 AC-NO PIC AA9999.
          05 MORE.
             10 STUFF PIC S99V9 OCCURS 2.
          05 SSN PIC 999/99/9999 VALUE '987-65-4321'.
       01 I BINARY PIC 99.
       PROCEDURE DIVISION.
       MAIN.
           MOVE 7.8 TO STUFF(1), MOVE -9 TO STUFF(2)
           JSON PARSE JSON-DATA INTO GRP
             NAME OF STUFF IS 'Value' SUPPRESS SSN

           DISPLAY "AC-NO:     " AC-NO.
           DISPLAY "STUFF(1): " STUFF(1).
           DISPLAY "STUFF(2): " STUFF(2).

           GOBACK.
**********
********** The example produces the following output:
********** AC-NO:    SX1234
********** STUFF(1): 7.8
********** STUFF(2): -9.0
```

MODIFY

Format 1

```
MODIFY {Control-Item       } [ ( {Index-1} ... ) ]
       {CONTROL AT Location}

  { { Property-Name          } [IS ] { [MULTIPLE] Property-Value [ LENGTH {IS} Length-
1 ] [GIVING Result-1] } } ...

    { PROPERTY Property-Type } [{ ARE }] { [TABLE   ]
                                 {  =  }
  { [NOT] Style-Name } ...
```

Location is defined as follows:

```
{ Screen-Loc [CELL  ]
             [CELLS ]
             [PIXEL ]
             [PIXELS]


  LINE NUMBER Line-Num [CELL  ]
                       [CELLS ]
                       [PIXEL ]
                       [PIXELS]

  {COLUMN  } NUMBER Col-Num [CELL  ]
  {COL     }                [CELLS ]
  {POSITION}                [PIXEL ]
  {POS     }                [PIXELS]

  }
```

Format 2

```
MODIFY {Window-Handle           }
       {WINDOW [Generic-Handle] }

  { Property-Name [IS ] Property-Value }
  {               [=  ]               } ...

  [ ON EXCEPTION Statement-1]

  [ NOT ON EXCEPTION Statement-2 ]

  [END-MODIFY]
```

Syntax rules

1. `CELL` and `CELLS` are synonymous.
2. `PIXEL` and `PIXELS` are synonymous.
3. `COLUMN`, `COL`, `POSITION` and `POS` are synonymous.
4. *Control-item* is a USAGE HANDLE data item or elementary Screen Section item that describes a control.
5. *Index-1* is a numeric expression. The parentheses surrounding *index-1* are required.
6. *Window-handle* is a USAGE HANDLE OF WINDOW or PIC X(10) data item.
7. *Generic-handle* is a USAGE HANDLE, HANDLE OF WINDOW or PIC X(10) data item.
8. *Screen-loc* is an integer data item or literal that contains exactly 4, 6, or 8 digits, or a group item of 4, 6, or 8 characters.
9. *Line-num*, *col-num* are numeric data items or literals. They can be non-integer values, except when pixels are specified.
10. *Length-1* is a numeric literal or data item. The LENGTH phrase may be specified only if the *value* or *property-value* immediately preceding it is an alphanumeric literal or data item, and not a figurative constant. In addition, the MULTIPLE option may not be specified along with the LENGTH phrase.
11. *Property-name* is the name of a property specific to the type of control being referenced. If the type of control is unknown to the compiler (as in a "DISPLAY OBJECT *object-1*" statement), then *property-name* may not be

used. You must use the PROPERTY *property-type* option instead.

12. *Property-type* is a numeric literal or data item. It identifies the property to modify. The numeric values that identify the various control properties can be found in the COPY library "iscontrols.def".
13. *Property-value* is a literal or data item. In the Procedure Division, *property-value* may also be a numeric expression (however, only the first *property-value* in a phrase may be an expression, subsequent values must be literals or data items). Note that the parentheses are required.

General rules

Format 1

1. A Format 1 MODIFY statement updates an existing control. *Control-item* should contain a handle returned by a DISPLAY *Control-Type* statement, or the name of an elementary Screen Section control item. If *control-item* does not refer to a valid control, the MODIFY statement returns the “Invalid Handle” error or has no effect if `iscobol.ignore_invalid_handle` is set to true. Note that controls referenced in the Screen Section are not valid until they have been created via a DISPLAY statement. If *control-item* refers to a valid control, the effect of the statement is to update the specified properties of the control and to redisplay it.
2. If *index-1* is specified, then certain properties in the control being modified are changed to match the value of *index-1*. This occurs before any modification occurs. The exact set of properties changed by the *index-1* depends on the control's type. Currently, two controls have properties that are changed in this way:

Control Type	Properties Affected
List Box	QUERY-INDEX
Grid	Y, X

   Each occurrence of *index-1* changes one property. The first occurrence changes the first property in the list presented in the preceding table. The second occurrence changes the second property.

   Supplying more index values than the control supports has no additional effect. You may omit trailing indexes; this leaves the corresponding properties unchanged.

   This feature can be used to simplify modification of specific elements of controls that hold multiple values. For example, you can modify the contents of row 2, column 3 in a grid with the statement:

```
MODIFY grid-1(2, 3), CELL-DATA = data-1
```

   This is equivalent to the statements:

```
MODIFY grid-1, Y = 2, X = 3
MODIFY grid-1, CELL-DATA = data-1
```

3. MODIFY simply locates the corresponding control and makes the specified modifications. This process does not examine any phrases specified in the Screen Section.

   By using the MODIFY verb, you do not need to specify an "item-to-add" property in the Screen Section

4. If the CONTROL phrase is used, the runtime modifies the control located at the screen position specified by the AT, LINE, and COLUMN phrases in the current window. A list of controls is maintained for each window. When attempting to modify a control at a specific location, the runtime searches this list, using the first control it finds that exactly matches the location. The list is maintained in the order that the controls are created. If the runtime does not find a control at the specified location, then the statement has no effect.

Note that a control cannot be moved with a MODIFY statement if it includes the CONTROL phrase. This is due to the fact that the AT, LINE, and COLUMN phrases are used to find the control instead of specifying its new position. To move a control, use the *control-handle* phrase. The compiler does not know the control-type specific style and property names. To Specify, use their definitions from the "iscontrols.def" COPY library.

5. The *style-name* phrase adds the named style to the control. If the NOT option is used with the *style-name* phrase, the named style is removed from the control instead. When a style is added, any conflicting styles are removed first. For example, if you add the FRAMED style to a button, then the UNFRAMED style is removed first.
6. When the LENGTH option is specified, *length-1* establishes the exact size of the value or *property-value*. The text value presented to the control may have no trailing spaces or may have trailing spaces added. When you specify the LENGTH option, the control uses exactly *length-1* characters of data with or without trailing spaces. However, when *length-1* is a value larger than the size of the data item it is modifying, then the size of the data item is used instead. If *length-1* is negative, it is ignored and the default handling occurs.
7. When properties return specific values, these values are placed in result-1 of the GIVING phrase. If the property does not have a pre-defined return value, result-1 is set to "1" if the property is set successfully, otherwise, *result-1* is set to "0". When a property is being given multiple values in a single assignment, as shown here,

   DISPLAY COLUMNS = (1, 10, 30)

   then *result-1* is set in response to the last value assigned. In the example above, *result-1* is set to 30. Because the meaning of each value depends on the property being set, you should consult the documentation on the specific property for the exact meaning.
8. The MODIFY verb takes a control's home position (upper left corner), its handle, the name of an elementary Screen Section item, or `^', as its first parameter. Only the properties of the control that are specified in the MODIFY statement are updated.

Format 2

9. A Format 2 MODIFY statement changes one or more attributes of an existing FLOATING or INITIAL WINDOW (not a subwindow). Attributes that are not specifically changed remain unchanged, except when a window is made larger, in which case it may also be repositioned in order to keep it on the screen. *Window-handle* or *generic-handle* identify the window to modify. If the WINDOW phrase is used and *generic-handle* is omitted, the current window is modified.
10. *Statement-1* executes if any part of the operation fails. An exception may be caused by one of the following situations:
    A. A window that has no input is activated.
    B. An external window error occurs. For example, the window does not exist or cannot be created for some reason.
    C. An illegal instruction is used.
11. *Statement-2* executes if the MODIFY statement succeeds.

Examples

Format 1 - Set the value of an entry-field from a variable

```
move "12345" to ws-cust-code
modify scr-cust-code value ws-cust-code
```

Format 1 - Modify the title of a label from a variable

```
move "Customer Address : " to ws-title
modify scr-cust-label title ws-title
```

Format 2 - Set the size in columns and lines of a graphical window

```
modify mywin-handle size ws-max-screen-size
                   lines ws-max-screen-lines
```

MOVE

Format 1

```
MOVE { Identifier-1 } TO { Identifier-2 } ... [END-MOVE]
     { Literal-1    }
```

Format 2

```
MOVE {CORRESPONDING} Identifier-3 TO Identifier-4 [END-MOVE]
     {CORR          }
```

Format 3

```
MOVE Identifier-1 TO Identifier-2 WITH {CONVERSION}
                                       {CONVERT   }

  [ ON EXCEPTION Statement-1 ]

  [ NOT ON EXCEPTION Statement-2 ]

[END-MOVE]
```

Format 4

```
MOVE { Class:>{method  ( [parameters] )} } TO Result-Item
             {field                     }
      ObjRef:>{method  ( [parameters] )}
             {field                     }
        SELF:>{method  ( [parameters] )}
             {field                     }
       SUPER:>{method  ( [parameters] )}
             {field                     }
        SELF
```

Format 5

```
MOVE POSITIONAL Group-1 TO Group-2 [ DELIMITED BY DEFAULT VALUE ]
```

Syntax rules

Format 1

1. If identifier-2 references a strongly-typed group item, identifier-1 shall be specified and be described as a group item of the same type.
2. Literal-1 and the data item referenced by identifier-1 are sending operands.
3. Identifier-2 is a receiving operand.
4. The TO keyword can be placed before each Identifier-2 when compiling with -cv.
5. The figurative constant SPACE shall not be moved to a numeric or numeric-edited item.
6. The figurative constant ZERO shall not be moved to an alphabetic data item.
7. The following table describes the validity of types of MOVE statements, specifies the validity of the move.

Category of sending operand		Category of receiving operand			
		Alphabetic	Alphanumeric-edited, Alphanumeric	National, National-edited	Numeric, Numeric-edited
Alphabetic		Yes	Yes	Yes	No
Alphanumeric		Yes	Yes	Yes	Yes
Alphanumeric-edited		Yes	Yes	Yes	No
National		Yes	Yes	Yes	Yes
National-edited		Yes	Yes	Yes	No
Numeric	Integer	No	Yes	Yes	Yes
	Noninteger	No	No	No	Yes
Numeric-edited		No	Yes	Yes	Yes

When moving National items to Alphabetic or Alphanumeric items, ensure that the destination item is large enough to store the national data according to the current encoding.

Format 2

8. The words CORR and CORRESPONDING are equivalent.
9. Identifier-3 and identifier-4 shall specify group data items and shall not be reference modified.
10. The corresponding data items within identifier-3 are sending operands. The corresponding data items within identifier-4 are receiving operands.

Format 5

11. Group-1 and Group-2 are group data items.

General rules

Format 1

1. 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.

Item identification for identifier-2 is performed immediately before the data is moved to the respective data item. If identifier-2 is a zero-length item, the MOVE statement leaves identifier-2 unchanged.

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. If identifier-1 is a zero-length item, it is as if literal-1 were specified as the figurative constant SPACE.

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.

2. Any move in which the sending operand is either a literal or an elementary item and the receiving item is an elementary item is an elementary move. Bit group items and national group items are treated as elementary items in the MOVE statement.

   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.

3. 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 deducting implied by, the receiving data item. When a national group item is referenced in a MOVE statement, no editing or de-editing takes place.

   NOTE - Bit group items and national group items are treated as elementary items in the MOVE statement.

   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. 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. 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.

   B. 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. The de-editing is performed as follows: the runtime retrieves digits from given

positions depending on the picture, digits at the '9' symbol positions and the sign are kept while digits at the editing character positions are discarded (for example, having the picture PIC 99/99 the runtime takes digits number 1, 2, 4 and 5, and discards digit number 3).
When the category of the sending operand is other than numeric-edited and the content of the sending operand would result in a false value in a numeric class condition, the results of the execution of the MOVE statement are undefined.

i. When a signed numeric item is the receiving item, the sign of the sending operand is placed in the receiving item. 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.

ii. 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.

iii. When the sending operand is described as alphanumeric or national, the sending operand is treated as if it were an unsigned integer of category numeric with the following characteristics:

   a. If the sending operand is a 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 31, in which case the rightmost 31 character positions are used.

   b. If the sending operand is a figurative constant, the number of digits is the same as the number of digits in 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 number of digits is the same as the number of characters in the literal. If the number of characters exceeds 31, the size of the sending operand is 31 digits and only the rightmost 31 alphanumeric characters in the literal are used.

4. 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 the following table.

   NOTE - MOVE of alphanumeric figurative constants to numeric items is an archaic feature and its use should be avoided.

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 — if usage is display — if usage is national	Alphanumeric Alphanumeric Alphanumeric National National Alphanumeric Alphanumeric National

SPACE, SPACES	Alphabetic Alphanumeric Alphanumeric-edited National National-edited	Alphanumeric Alphanumeric Alphanumeric 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		

Note - When ALL is used, if the receiving item is national, the literal should be escaped, e.g.

```
MOVE ALL N'B' TO PIECE
```

Format 2

5. Data items within identifier-3 are selected to be moved to selected data items within identifier-4 according to the rules specified in the CORRESPONDING phrase. The results are the same as if the user had referred to each pair of corresponding identifiers in separate MOVE statements except that if subscripting is specified for identifier-3 or identifier-4, the subscript value used is that resulting from the evaluation of the subscript at the start of the execution of the statement.

   NOTE - For purposes of MOVE CORRESPONDING, bit group items and national group items are processed as group items, rather than as elementary items.

Format 3

6. A Format 3 MOVE statement performs a logical conversion of Identifier-1into the format of Identifier-2. Identifier-2 may be any type of data item. This is normally done to convert a character representation of a number into the corresponding numeric value. The rules of conversion are the same as the rules used by the CONVERT option of the ACCEPT statement.
7. If the ON EXCEPTION phrase is specified, then statement-1 executes when a conversion error occurs. If a conversion error occurs, then the value assigned to Identifier-2 is the value determined by ignoring the illegal characters in Identifier-1. If the NOT ON EXCEPTION phrase is specified, then statement-2 executes when no conversion error occurs.

Format 4

8. A Format 4 MOVE statement moves the result of a method invokation in object oriented programming.
9. Result-Item should be defined according to the result of the method and the field type. To receive strings and numbers, Result-Item can be a standard COBOL data-item with picture PIC X(n) or PIC 9(n). To receive an instance of an object, then Result-Item should be defined as OBJECT REFERENCE to that object.
10. If the field is a COBOL data item, it must be indicated upper case with hyhpens replaced by underscores, that is the way isCOBOL internally defines data items. For example, having `77 my-item pic x`, you will reference it using `SELF:>MY_ITEM` and not `SELF:>my-item`.
11. SELF means the current class. It can be used in a OBJECT paragraph. In a FACTORY paragraph the class logical

name must be used in order to reference the current class.

```
IDENTIFICATION DIVISION.
CLASS-ID. MY_OBJ AS "MyObj".

IDENTIFICATION DIVISION.
OBJECT.

DATA DIVISION.
WORKING-STORAGE SECTION.
PROTECTED.
77 Item-1 PIC X(3).

PROCEDURE DIVISION.
IDENTIFICATION DIVISION.
METHOD-ID. TEST_SELF AS "TestSelf".
PROCEDURE DIVISION.
MAIN.
    SELF:>method1().
    DISPLAY SELF:>ITEM_1.
END METHOD.

IDENTIFICATION DIVISION.
METHOD-ID. METHOD1 as "method1".
PROCEDURE DIVISION.
MAIN.
    CONTINUE.
END METHOD.

END OBJECT.
```

```
IDENTIFICATION DIVISION.
CLASS-ID. MY_CLASS AS "MyClass".

IDENTIFICATION DIVISION.
FACTORY.

DATA DIVISION.
WORKING-STORAGE SECTION.
PROTECTED.
77 Item-1 PIC X(3).

PROCEDURE DIVISION.
IDENTIFICATION DIVISION.
METHOD-ID. TEST_SELF AS "TestSelf".
PROCEDURE DIVISION.
MAIN.
    MY_CLASS:>method1().
    DISPLAY MY_CLASS:>ITEM_1.
END METHOD.

IDENTIFICATION DIVISION.
METHOD-ID. METHOD1 as "method1".
PROCEDURE DIVISION.
MAIN.
    CONTINUE.
END METHOD.

END FACTORY.
```

12. SUPER refers to the class that the current class is inheriting from, if any.
13. Only PROTECTED fields can be referenced from FACTORY paragraphs and superclasses.
14. Moving SELF to Result-Item allows you to retrieve the instance of the current class.

Format 5

15. A Format 5 MOVE statement moves every sub item in Group-1 to the sub item in Group-2 that has the same ordinal position. The first item in Group-1 is moved to the first item in Group-2, the second item in Group-1 is moved to the second item in Group-2, and so on.
16. RENAMES and REDEFINES fields are skipped.

For example, the following code

```
working-storage section.
...
01 a.
  02 b pic x(10).
  02 b-red redefines b pic 9(10).
  02 c pic x(10).

01 a2.
   02 b2 pic x(10).
   02 c2 pic x(10).
   02 c2-red redefines c2 pic 9(10).

procedure division.
...
    move positional a to a2.
```

is equivalent to

```
working-storage section.
...
01 a.
  02 b pic x(10).
  02 b-red redefines b pic 9(10).
  02 c pic x(10).

01 a2.
   02 b2 pic x(10).
   02 c2 pic x(10).
   02 c2-red redefines c2 pic 9(10).

procedure division.
...
    move b to b2.
    move c to c2.
```

17. When Group-1 includes a fixed capacity table and the corresponding item in Group-2 is a dynamic capacity table, the Runtime moves all the items in the table, unless the DELIMITED BY DEFAULT VALUE clause is used. In such case, the Runtime moves all the items in the table until it finds an item whose value matches the default value of the dynamic capacity table. For a correct result, all the items in the dynamic capacity table should have a VALUE clause and the dynamic capacity table must specify the INITIALIZED clause.

For example

```
       working-storage section.
       ...
       01 struct-1.
          03 occ-1 occurs 100.
             05 struct-1-var-1 pic x(10).

       01 struct-2.
          03 occ-2 occurs dynamic capacity cap initialized.
             05 struct-2-var-1 pic x(10) value "a".

       procedure division.
       ...
           move 1 to struct-1-var-1(1).
           move "a" to struct-1-var-1(2).
           move positional struct-1 to struct-2 delimited by default.
```

In the above case, the Runtime will move only the first item of occ-1 to occ-2, and occ-2 will have a capacity of 1 after the MOVE. If the program didn't move "a" to struct-1-var-1(2), instead, the Runtime would move all the 100 items of occ-1 to occ-2, and occ-2 would have a capacity of 100 after the MOVE.

Examples

Format 1 - Move literals or variables to variables

```
move 93.5 to num-1    *> num-1 could be pic 9(2)V9(2)
move "hello" to str-1 *> str-1 could be pic x(10)
move num-1 to num-2
move str-1 to str-2 str-3 str-4 *> More than one variables receive the value
move zeroes to num-1 num-2 num3
```

Format 2 - Move group item to group item with corresponding children items

```
01 work-hours.
   05 test-hours    pic 9(3) value 10.
   05 doc-hours     pic 9(3) value 11.
01 total-work-hours.
   05 doc-hours     pic 9(3) value 4.
   05 support-hours pic 9(3) value 2.
   05 test-hours    pic 9(3) value 2.

...

move corresponding work-hours to total-work-hours
*> doc-hours and test-hours of total-work-
hours will end up having the same values as in work-hours
```

Format 3 - Move with conversion using on exception clause

```
move "123.456" to str-1               *> str-1 could be pic x(10)
move str-1 to num-1 with conversion   *> num-1 could be pic 9(3)v9(3)
     on exception display message "Move with conversion failed!"
     not on exception display message "Good move"
end-move
*> The message displayed will be : Good move
```

Format 3 - Move with conversion using on exception clause

```
move "12X.B56" to str-1                *> str-1 could be pic x(10)
move str-1 to num-1 with conversion    *> num-1 could be pic 9(3)v9(3)
     on exception display message "Move with conversion failed!"
     not on exception display message "Good move"
end-move
*> The message displayed will be : Move with conversion failed!
*> However, the value of num-1 will be : 12.56
```

Format 3 - Plain move with conversion

```
move "123.4" to str-1                 *> str-1 could pib x(10)
move num-1 to str-1 with conversion   *> num-1 could pic 9(3)v9(1)
```

Format 4 - Invoke methods of a Java Class to get System Information

```
       configuration section.
       repository.
           class Sys as "java.lang.System".

       working-storage section.
       77  buffer      pic x(50).

       procedure division.
       main.
           move Sys::>getProperty("os.name") to buffer
           display "Operating System: " buffer.

           move Sys::>getProperty("os.arch") to buffer
           display "OS Architecture: " buffer.

           move Sys::>getProperty("java.version") to buffer
           display "Java Version: " buffer.
```

Format 5 - Move complex structures with tables using a single statement

```
       working-storage section.
       01 struct-fix.
          03 g1-name pic x(50).
          03 g1-table occurs 100.
             05 g1-account-id pic 9(3) value 0.
             05 g1-account-short-des pic x(20) value space.
             05 g1-account-notes pic x(1000) value space.
          03 g1-address pic x(50).

       01 struct-dyn.
          03 g2-name pic x any length.
          03 g2-table occurs dynamic
                           capacity cap-g2-table-occ
                           initialized.
             05 g2-account-id pic 9(3) value 0.
             05 g2-account-short-des pic x(20) value space.
             05 g2-account-short-des pic x any length value space.
          03 g2-address pic x any length.

       procedure division.
       main.
           move positional struct-fix to struct-dyn
```

MULTIPLY

Format 1

```
MULTIPLY {Identifier-1} BY { Identifier-2 [ROUNDED]  } ...
         {Literal-1   }
         {Class-1     }

  [ ON SIZE ERROR Imperative-Statement-1 ]

  [ NOT ON SIZE ERROR Imperative-Statement-2 ]

  [END-MULTIPLY]
```

Format 2

```
MULTIPLY {Identifier-1} BY {Identifier-2}
         {Literal-1   }    {Literal-2   }
         {Class-1     }    {Class-2     }

  GIVING { Identifier-3 [ROUNDED] } ...

  [ ON SIZE ERROR Imperative-Statement-1 ]

  [ NOT ON SIZE ERROR Imperative-Statement-2 ]

  [END-MULTIPLY]
```

Syntax rules

1. Identifier-1 and identifier-2 shall reference a data item of category numeric.
2. Identifier-3 shall reference a data item of category numeric or numeric-edited.
3. Literal-1 and literal-2 shall be numeric literals.
4. Class-1 and Class-2 shall be OBJECT-REFERENCE of Java numeric primitive types (i.e. "int") or objects (i.e. "java.lang.Integer")

General rules

1. When format 1 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. 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.

Examples

Format 1 - Multiply literal or variable by variable

```
move 20   to ws-factor
move 10  to ws-age
multiply 2 by ws-age
*> resulting value of ws-age : 20
multiply ws-factor by ws-age
         on size error display message "size error"
end-multiply
*> resulting value of ws-age : 400
*> if ws-age is only pic 9(2) the "size error" message will be displayed

move 10 to ws-age
move 20 to ws-age2
move 15 to ws-age3
multiply 2 by ws-age ws-age2
*> resulting values: ws-age = 20 , ws-age2 = 40, ws-age3 = 30
```

Format 1 - Rounding the result of the multiply

```
move     5.99     to ws-radius
multiply 3.141593 by ws-radius rounded
*> If ws-radius is pic 9(3)v9(3), result will be : 18.818
*> If ws-radius is pic 9(3)v9(2), result will be : 18.82
```

Format 2 - Rounding the result of the multiply and placing result on different variable

```
move     5.99     to ws-radius
multiply 3.141593 by ws-radius rounded
  giving ws-result
end-multiply
*> If ws-result is pic 9(3)v9(3), result will be : 18.818
*> If ws-result is pic 9(3)v9(2), result will be : 18.82
```

NEXT SENTENCE

General Format

```
NEXT SENTENCE
```

Syntax Rules

1. A NEXT SENTENCE statement is allowed anywhere if either -ca, -cm or -cr compiler flags are used. Without these flags, a NEXT SENTENCE statement is allowed only anywhere a conditional statement is allowed.

General Rules

1. A NEXT SENTENCE statement transfers the flow of execution to the logically next COBOL verb following the next period.

Examples

Continue execution after the if statement

```
if 1 = 1
   next sentence
else
   display "this cannot ever be displayed"
end-if.  *> The period here is mandatory for the next
         *> sentence to know where to continue
display "hello".
```

NOTE

General Format

```
NOTE text
```

Syntax Rules

1. Text can be any combination of characters from the computer character set.

General Rules

1. If the NOTE statement is the first sentence of a paragraph, it must not be followed by any text and it corresponds to an EXIT PARAGRAPH statement.
2. If a NOTE statement appears as other than the first sentence of a paragraph, text up to the next separator period is treated as comment.

Note - this statement is supported only with the -cv compiler flag.

Examples

Having a paragraph, p1, with a note only as contents, having another paragraph, p2, with a note in the middle

```
procedure division.
main.
  perform p1
  perform p2
  goback.

p1.
  note
  display message "This message will not appear"
  .

p2.
  display message "Entering p2".
  note This is commentary
  until the next period
  is found.
  display message "Leaving p2"
  .
```

ON

General Format

```
ON { literal-1    } [ AND EVERY { literal-2    } ] [ UNTIL { literal-3    } ]
   { identifier-1 }             { identifier-2 }           { identifier-3 }

{ imperative-statement-1 }
{ NEXT SENTENCE          }

[ { ELSE      }   { imperative-statement-2 } ]
  { OTHERWISE }   { NEXT SENTENCE          }
```

Syntax rules

1. Identifier-1, identifier-2 and identifier-3 must describe unsigned integer numeric elementary items.
2. Literal-1, literal-2 and literal-3 must be unsigned numeric literals.

General rules

1. Prior to the first execution of each ON statement, a counter is implicitly defined for that ON statement and is initialized to be zero.
2. Identifier-1, identifier-2 and identifier-3 should, if specified, contain positive integer values at the time of execution of the ON statement. Varying these values between executions of the ON statement will affect subsequent executions of the ON statements.
3. The implicit ON counter cannot be affected in any way other than by transfer of execution flow to that ON statement. Execution of the EXIT PROGRAM statement and subsequent CALL of the program without intervening CANCEL has no effect upon the implicit ON counter value; the ON counter of a called program can only be reset by the canceling of that program.
4. The following value-list is then evaluated:
   A. The current value of identifier-1 or literal-1,
   B. A sequence of values being the results of repeatedly adding the current value of identifier-2 or literal-2 to the current value of identifier-1 or literal-1 until the value of identifier-3 or literal-3 is reached.
5. The implicit-ON-counter is then compared with each of this list of values. If an equality is found, then imperative-statement-1 is executed. If no equality is found, then imperative-statement-2, if specified, is executed.
6. A severe error is returned if NEXT SENTENCE is in the THEN block and is not followed either by ELSE or dot or if it is in the ELSE block and is not followed by dot. The error is not returned under the -ca compile option, the -cm compile option and the -cr compile option, because with these options NEXT SENTENCE is treated as a normal statement.

Examples

Display a message the first time the 'elab' paragraph is executed

```
main.
     perform elab 10 times.
elab.
     on 1 display "Loop started".
     ...
```

OPEN

General Format

```
OPEN [ EXCLUSIVE ]

  { {INPUT } { [Sharing-Phrase] { File-name-1 [ WITH NO REWIND ] } ... } ...
    {OUTPUT} { File-name-1 [Lock-Option] (only with -CA option)  }
    {I-O   }
    {EXTEND}
```

Where *Sharing-Phrase* is:

```
  SHARING WITH { ALL OTHER }
               { NO OTHER  }
               { READ ONLY }
```

Where *Lock-Option* is:

```
  { ALLOWING { NO OTHERS } }
             { OTHERS   }
             { READERS   }
             { WRITERS   }
             { UPDATERS  }
             { ALL       }

  { {WITH} {LOCK       }    }
    {FOR } {MASS-UPDATE}
           {BULK-ADDITION}
```

Syntax rules

1. The OPEN statement for a report file shall not contain the INPUT phrase or the I-O phrase.
2. The EXTEND phrase shall be specified only if the access mode of the file connector referenced by file-name-1 is sequential.
3. The files referenced in the OPEN statement need not all have the same organization or access.
4. The NO REWIND phrase may be specified only for sequential files.
5. The NO REWIND phrase may be specified only when the INPUT or OUTPUT phrase is specified.
6. If the SHARING phrase is omitted from the OPEN statement and the ALL phrase is specified in the SHARING clause of the file control entry for file-name-1 or if the ALL phrase is specified on the OPEN statement, the LOCK MODE clause shall be specified in the file control entry for file-name-1.
7. The I-O phase shall not be specified if the FORMAT clause is specified in the file description entry for file-name-1.

General rules

1. The file connector referenced by file-name-1 shall not be open. If it is open, the execution of the OPEN statement is unsuccessful and the I-O status associated with file-name-1 is set based on your iscobol.file.status * setting.
2. The successful execution of an OPEN statement associates the file connector referenced by file-name-1 with a physical file if the physical file is available, and sets the open mode of the file connector to input, output, I-O, or extend, depending on the keywords INPUT, OUTPUT, I-O or EXTEND specified in the OPEN statement. The

open mode determines the input-output statements that are allowed to reference the file connector as shown in the following table.

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

A physical file is available if it is physically present and is recognized by the operating environment. The table above shows the results of opening available and unavailable physical files that are not currently open by another file connector. The table below shows the results of opening available physical files that are currently open by another file connector, including those implicitly opened by the SORT and MERGE statements.

Open request		**Most restrictive existing sharing mode and open mode**				
		sharing with no other	**sharing with read only**		**sharing with all other**	
		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

3. The successful execution of an OPEN statement makes the associated record area available to the runtime element. 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.
4. When a file connector is not open, no statement shall be executed that references the associated file-name, either explicitly or implicitly, except for a MERGE or SORT statement with the USING or GIVING phrase, or an OPEN statement.
5. The OPEN statement for a report file connector shall be executed before the execution of an INITIATE statement that references a report-name that is associated with file-name-1.
6. For a given file connector, an OPEN statement shall be successfully executed prior to the execution of any other permissible input-output statement. In the table below, an '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.

Access mode	**Statement**	**Open mode**			
		Input	**Output**	**I-O**	**Extend**
Sequential	READ	X		X	
	WRITE		X		X
	REWRITE			X	
	START	X		X	
Sequential (relative and indexed files only)	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	

7. Execution of the OPEN statement does not obtain or release the first record.
8. During the execution of the OPEN statement when the file connector is matched with the physical file and the physical file exists, the attributes of the file connector as specified in the file control paragraph and the file description entry are compared with the fixed file attributes of the physical file. If the attributes don't match, a file attribute conflict condition occurs, the execution of the OPEN statement is unsuccessful, and the I-O status associated with file-name-1 is set based on your iscobol.file.status * setting. The implementor defines which of the fixed-file attributes are validated during the execution of the OPEN statement. The validation of fixed-file attributes may vary depending on the organization of the file.
9. If the physical file is not present, and the INPUT phrase is specified in the OPEN statement, and the OPTIONAL clause is specified in the file control entry for file-name-1, the file position indicator in the file connector referenced by file-name-1 is set to indicate that an optional input file is not present.
10. When the organization of the file referenced by file-name-1 is sequential or relative and the INPUT or I-O phrase is specified in the OPEN statement, the file position indicator for that file connector is set to 1. When the organization is indexed, 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.
11. 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.
12. If the I-O phrase is specified, the physical file shall support the input and output statements that are permitted for the organization of that file when opened in the I-O mode. If the physical file does not support those statements, the I-O status value for file-name-1 is set based on your iscobol.file.status * setting and the execution of the OPEN statement is unsuccessful. The successful execution of an OPEN statement with the I-O phrase sets the open mode of file connector referenced by file-name-1 to open in the I-O mode.
13. If the physical file is not present, and the EXTEND or I-O phrase is specified in the OPEN statement, and the OPTIONAL clause is specified in the file control entry for file-name-1, the OPEN statement 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 element.
14. If the OUTPUT phrase is specified, the successful execution of the OPEN statement creates the physical file. After the creation of the physical file, the file contains no records. If physical pages have meaning for the physical file, the positioning of the output medium with respect to physical page boundaries is implementordefined following the successful execution of the OPEN statement.
15. Upon successful execution of the OPEN statement, the current volume pointer is set:
    A. To point to the first or only reel/unit in the physical file if INPUT or I-O is specified.
    B. To point to the reel/unit containing the last record in the physical file if EXTEND is specified.

C. To point to the newly created reel/unit in the physical file for an unavailable file if EXTEND, I-O, or OUTPUT is specified.

16. 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, the sharing-phrase, retry-phrase, and REWIND phrase as specified in the OPEN statement. If an implicit OPEN statement results in the execution of a declarative procedure that executes a RESUME statement with the NEXT STATEMENT phrase, processing resumes at the next implicit OPEN statement, if any.
17. The SHARING phrase is effective only for files that are shareable.
18. The SHARING phrase specifies the level of sharing permitted for the physical file associated with file-name-1 and specifies the operations that may be performed on the physical file through other file connectors sharing the physical file.
19. 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, the implementor shall define the sharing mode that is established for each file connector.
20. The ALLOWING phrase is similar to SHARING phrase. It is supported by the compiler only with –ca compiler option.
    A. The ALLOWING phrase permits the following types of file locking: ALLOWING ALL, ALLOWING READERS, and ALLOWING NO OTHERS. File locking is enforced for all file types residing on disk but may not be enforced for non-disk files for some operating systems.
    B. The following phrases imply the ALLOWING ALL form of file locking:

        i. No lock-option specified
        ii. ALLOWING ALL
        iii. ALLOWING WRITERS
        iv. ALLOWING UPDATERS

        This file locking mode indicates that other programs can access the file without restriction except that another program may not execute an OPEN OUTPUT while this program keeps the file open
    C. The following phrases imply the ALLOWING READERS form of file locking:

        i. ALLOWING READERS
        ii. EXCLUSIVE or WITH LOCK specified with the INPUT phrase

        A file open in this mode does not allow any other program to open this file other than with the INPUT phrase. Also, this OPEN will fail if any other programs currently have the file open unless the INPUT phrase was used by all of these other programs
    D. These phrases imply the ALLOWING NO OTHERS form of file locking:

        i. ALLOWING NO OTHERS
        ii. EXCLUSIVE or WITH LOCK specified with the OUTPUT, I-O, or EXTEND phrases
        iii. WITH MASS-UPDATE
        iv. WITH BULK-ADDITION

        This form of file locking does not allow any other programs to open the file, and this OPEN will fail if any other programs currently have the file open.
21. If the file connector referenced by file-name-1 is locked by a previously-executed CLOSE statement with the LOCK phrase, the execution of the OPEN statement is unsuccessful and the I-O status associated with that file connector is set based on your iscobol.file.status * setting.

22. If the execution of the OPEN statement is unsuccessful, the physical file is not affected and the value is placed in the I-O status associated with file-name to indicate the condition that caused the OPEN statement to be unsuccessful.
23. The WITH MASS-UPDATE phrase indicates to the runtime system that the file will be heavily updated by the program. The runtime system may be able to use this information to access the file more efficiently.
24. The WITH BULK-ADDITION is similar to WITH MASS-UPDATE. It behaves differently depending on the file handler.
    A. JIsam treats BULK-ADDITION the same way as MASS-UPDATE
    B. c-tree supports BULK-ADDITION from version 9.5.46192 and manages it by creating indexes at the close of the file. Unique key violations are traced in the c-tree server log file without triggering Declaratives.

Examples

Open several files for input, output and i-o

```
open input  customers invoices purchase-orders
open output rep-customers rep-invoices
open i-o    invoice-detail
```

Open files and lock them until they get closed

```
open i-o   customers with lock
open input invoices with lock
```

Open files locking them for massive update or addition

```
open i-o  pay-receipts     for mass-update
open i-o  pending-invoices for bulk-addition
```

PERFORM

Format 1

```
PERFORM  [IN THREAD] procedure-name-1 [ {THROUGH} procedure-name-2 ]
                                       {THRU   }
         [HANDLE IN Handle-1] [times-phrase  ]
                   [until-phrase  ]
                   [varying-phrase]
```

Format 2

```
PERFORM [times-phrase  ] imperative-statement-1 END-PERFORM
        [until-phrase  ]
        [varying-phrase]
```

where times-phrase is:

```
{identifier-1} TIMES
{integer-1   }
```

where until-phrase is:

```
[ WITH TEST {BEFORE} ] UNTIL {condition-1}
            {AFTER }
```

where varying-phrase is:

```
[ WITH TEST {BEFORE}
            {AFTER }

VARYING {identifier-2} FROM   {identifier-3} [BY {identifier-4}] UNTIL condition-1
        {index-name-1}        {index-name-2}     {literal-2   }
        {inline-declaration-1}{literal-1   }

[ AFTER {identifier-5} FROM   {identifier-6} [BY {identifier-7}] UNTIL condition-2]...
        {index-name-3}        {index-name-4}     {literal-4   }
        {inline-declaration-2}{literal-3   }
```

Format 3

```
PERFORM  imperative-statement-1 UNTIL EXIT END-PERFORM
```

Syntax rules

1. If neither the TEST BEFORE nor the TEST AFTER phrase is specified, the TEST BEFORE phrase is assumed.
2. Each identifier shall reference a numeric elementary item described in the data division. Identifier-1 shall be an integer.
3. Each literal shall be numeric.
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 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. The literal in the BY phrase shall not be zero.
8. Condition-1, condition-2, ... , may be any conditional expression.
9. Procedure-name-1 shall be the name of either a paragraph or a section in the same source element as that in which the PERFORM statement is specified.
10. Procedure-name-2 shall be the name of either a paragraph or a section in the same source element as that in which the PERFORM statement is specified.
11. Inline-declaration-1 and inline-declaration-2 define primitive numeric items that will exist only during the PERFORM cycle. The syntax is

```
identifier as type
```

where *type* can be one of the following Java primitive types to be specified between quotes: double, float, int, long or short.
Being limited to a specific PERFORM cycle the same identifier can be used in sibling PERFORM cycles, but not in nested PERFORM cycles.

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.
2. An inline PERFORM statement and an out-of-line PERFORM statement function identically according to the following rules. For an out-of-line PERFORM statement, the specified set of statements consists of all statements beginning with the first statement of procedure-name-1 and ending with the last statement of procedure-name-2, or, if procedure-name-2 is not specified, the last statement of procedure-name-1. For an inline PERFORM statement, the specified set of statements consists of all statements contained within the PERFORM statement.
3. 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 7, 8, and 9. 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-2 is not specified, the return mechanism is after the last statement of procedure-name-1.
   B. If procedure-name-2 is specified, the return mechanism is after the last statement of procedure-name-2.
4. 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.

   NOTE - Statements such as the GO TO statement, the PERFORM statement, and the procedure format of the EXIT statement may occur in the flow of execution of the specified set of statements, however the flow of execution should eventually pass to the end of procedure-name-2.
5. 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.
6. 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.
7. If times-phrase is specified, the specified set of statements is performed the number of times specified by integer-1 or by the value of the data item referenced by identifier-1 at the start of the execution of the PERFORM statement. If at the start 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.

   NOTE - During execution of the PERFORM statement, a change to the contents of identifier-1 does not alter the number of times the specified set of statements is performed.
8. 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 the specified set of statements 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. Item identification associated with the operands specified in condition-1 is done each time the condition is tested.

9. If varying-phrase is specified, the execution of the PERFORM statement augments the data items referenced by one or more identifiers or the indexes referenced by one or more index-names in an orderly fashion. In the following rules, the data items referenced by identifier-2 and identifier-5 and the indexes referenced by index-name-1 and index-name-3 are referred to as the induction variables. The content of the data item referenced by the identifier, the occurrence number corresponding to the value of the index referenced by the index-name, or the value of the literal referenced in the FROM phrase is referred to as the initialization value. The content of the data item referenced by the identifier or the value of the literal in a BY phrase is referred to as the augment value. For any BY phrase that is omitted, the augment value is 1. Item identification for identifier-2, identifier-5, index-name-1, or index-name-3 is done each time the content of the data item referenced by the identifier or the value of the index referenced by the index-name is set or augmented. Item identification for identifier-3, identifier-4, identifier-6, identifier-7, index-name-2, and index-name-4 is done each time the content of the data item referenced by the identifier or the index referenced by the index-name is used in a setting or augmenting operation. Item identification associated with the operands specified in condition-1 or condition-2 is done each time the condition is tested.

   NOTE - If an augment value is less than 0, the induction variable is actually decremented by the absolute value of the augment value.

   The sequence of operation of the PERFORM statement is as follows:

   A. All induction variables are set to their associated initialization values in the left-to-right order in which the induction variables are specified.
   B. If the TEST AFTER phrase is specified, and there is no AFTER phrase, the specified set of statements is executed once and condition-1 is tested. If the condition is false, the induction variable is incremented by the augment value, and the specified set of statements is executed again. The cycle continues until condition-1 is tested and found to be true, at which point control is transferred to the end of the PERFORM statement. At that point, the induction variable contains the value it contained at the completion of the execution of the specified set of statements.
   C. If the TEST AFTER phrase is specified, and there is one or more AFTER phrase, the following occurs:

      i. The specified set of statements is executed.
      ii. The rightmost condition-2 is then evaluated.
      iii. If the rightmost condition-2 is false, the associated induction variable is incremented by the associated augment value, and execution proceeds with step a.
      iv. If the last condition evaluated is true, the condition to its left is evaluated. This is repeated until either a false condition is found or the last condition evaluated is condition-1 and condition-1 is true. If a false condition is found, the induction variable associated with that condition is incremented by the associated augment value, all induction variables to the right of the false condition are set to their initialization values, and execution proceeds with step a. If no condition is found to be false, control is transferred to the end of the PERFORM statement.

   D. If the TEST AFTER phrase is not specified and there is no AFTER phrase, condition-1 is evaluated, and if it is true, control is transferred to the end of the PERFORM statement. If it is false, the specified set of statements is executed. Then, the induction variable is incremented by the augment value, and condition-1 is evaluated again. When control is passed to the end of the PERFORM statement, the induction variable contains the value it contained when condition-1 was evaluated.
   E. If the TEST AFTER phrase is not specified, and there is one or more AFTER phrase, the following occurs:

      i. Condition-1 is evaluated.

If condition-1 is true, control is transferred to the end of the PERFORM statement; otherwise, the condition-2 immediately to the right becomes the current condition.

ii. The current condition is evaluated.

If the current condition is true:

a. the induction variable associated with the current condition is set to its initialization value

b. the condition to the left of the current condition becomes the current condition

c. the induction variable associated with the new current condition is incremented by its associated augment value

d. if the current condition is condition-1 execution proceeds to step a, else execution proceeds to the beginning of step b.

otherwise:

a. if there is another AFTER phrase to the right of the current condition,

—the condition associated with that AFTER phrase becomes the current condition

—execution proceeds to the beginning of step b;

b. otherwise

—the specified set of statements is executed

—the induction variable associated with the current condition is incremented by the augment value

—execution proceeds to the beginning of step b.

During the execution of the specified set of statements associated with the PERFORM statement, all changes to the induction variable, the variables associated with the augment value, and the variables associated with the initialization value have immediate effect and all subsequent references to the associated data items use the updated contents.

10. 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 in the range of the PERFORM statement, except for statements executed as the result of a transfer of control by an EXIT METHOD, EXIT PROGRAM, or GOBACK statement specified in the same instance of the same source element as the PERFORM statement. Declarative procedures that are executed as a result of the execution of statements in the range of a PERFORM statement are included in the range of the PERFORM statement. The statements in the range of a PERFORM statement need not appear consecutively in the source element.
11. If handle-1 is specified, the new thread's unique ID is stored in handle-1
12. When the THREAD option is used, a new thread is created by the PERFORM statement. Once control returns to the end of the PERFORM statement, the thread is terminated. All of the statements contained in the scope of the PERFORM are executed in the new thread.
13. Data items defined with an inline declaration exist only for the specific PERFORM cycle where they were defined and are disposed as soon as the cycle terminates. This kind of data item can’t be edited or monitored by the Debugger.

Format 3

14. The set of statements is performed until an EXIT PERFORM statement is executed.

Examples

Format 1 - Perform paragraphs in separate thread

```
working-storage section.
77 t-handle-1 usage handle of thread.

procedure division.
main.
  perform thread par-1 handle t-handle-1
  display message "This message displays even before par-1 is done"
  ...
```

Format 2 - Perform paragraphs or statements n times

```
perform par-1 thru par-5 10 times
perform par-2 5 times
perform 10 times
  add 1 to var-1
end-perform
```

Format 2 - Perform paragraphs or statements until condition evaluates true

```
perform multiply-var-1 until var-1 > 900
perform compute-interest thru compute-exit until var-interest > 0
perform until var-1 > 900
  multiply 2 by var-1
end-perform
```

Format 2 - Perform paragraphs or statements varying variable

```
perform count-items varying ws-count from 1 by 1 until ws-item-count > 9850
perform until varying ws-count from 1 by 1 until var-1 > 900
  multiply ws-count by var-1
end-perform
```

Perform paragraphs or statements testing the condition after the first iteration

```
perform multiply-var-1 test after until var-1 > 900
perform compute-interest thru compute-exit test after until var-interest > 0
perform test after until var-1 > 900
  multiply 2 by var-1
end-perform
```

RAISE

General Format

```
RAISE Exception-Class
```

Syntax Rules

1. In a PROGRAM-ID program, Exception-Class is a java.lang.RuntimeException class or any known subclass.
2. In a CLASS-ID program, Exception-Class is a java.lang.Exception class or any known subclass. If Exception-Class is a superclass of java.lang.RuntimExcеption, then the method in which the RAISE statement is used, must specify Exception-Class in the RAISING clause of the PROCEDURE DIVISION.

Note - if the above rules are not respected, a incompilable Java source may be generated by the isCOBOL Compiler.

General Rules

1. If caught, the EXCEPTION-OBJECT will be set to the instance of the Exception-Class

Examples

Perform three jobs in sequence and stop at the first one that fails.

```
 program-id. threeJobs.
 configuration section.
 repository.
    class runex as "java.lang.RuntimeException".
 working-storage section.
 77 elab-status pic 9.
    88 ok       value 1.
    88 failed   value 0.
 procedure division.
 MAIN.
    try
       perform JOB-1
       perform JOB-2
       perform JOB-3
       display "All jobs completed successfully!"
    catch exception
       display exception-object:>getMessage()
    end-try.
    goback.
 JOB-1.
*>  job logic here
    if failed
       raise runex:>new("Job 1: failed")
    end-if.
 JOB-2.
*>  job logic here
    if failed
       raise runex:>new("Job 2: failed")
    end-if.
 JOB-3.
*>  job logic here
    if failed
       raise runex:>new("Job 3: failed")
    end-if.
```

Invoke a CLASS-ID that may return a custom exception named BadParamException:

BadParamException.cbl

```
identification division.
class-id. BadParamException as "BadParamException" inherits JException.
configuration section.
repository.
class JException as "java.lang.Exception"
class JString as "java.lang.String".
identification division.
object.
procedure division.
identification division.
method-id. new as "new".
procedure division.
main.
    super:>new()
end method.
identification division.
method-id. new as "new".
linkage section.
    77 lk-message object reference JString.
procedure division using lk-message.
main.
    super:>new(lk-message)
end method.
end object.
```

StrProcessor.cbl

```
identification division.
class-id. StrProcessor as "StrProcessor".
configuration section.
repository.
class myEx as "BadParamException"
class JString as "java.lang.String".
identification division.
factory.
procedure division.
identification division.
method-id. processString as "processString".
working-storage section.
77 wk-str pic x any length.
77 i pic 99.
linkage section.
77 lk-str object reference JString.
procedure division using lk-str raising myEx.
main.
    set wk-str to lk-str.
*> string must be at least 5 characters in size
    if function length(wk-str) < 5
       raise myEx:>new("string is less than 5 characters in size")
    end-if.
*> string must not contain spaces
    initialize i.
    inspect wk-str tallying i for all spaces
    if i > 0
       raise myEx:>new("string includes spaces")
    end-if.
*> go ahead processing the string
end method.
end factory.
```

PROG.cbl

```
program-id. prog.
configuration section.
repository.
class sp as "StrProcessor".
procedure division.
main.
    try
       sp:>processString("X Y Z")
*>the above statement will raise a BadParamException as the string includes spaces
    catch exception
       exception-object:>printStackTrace()
    end-try.
    goback.
```

READ

Format 1

```
READ File-Name-1  [NEXT     ] RECORD
                  [PREVIOUS]
                  [BACKWARD]

  [ WITH [ { NO     } ] LOCK ]
         { KEPT   }
         { IGNORE }
         { WAIT   }

  [ ALLOWING UPDATERS ]

  [ INTO Identifier-1 ]

  [ SIZE Identifier-2 ]

  [ AT END Imperative-Statement-1 ]

  [ NOT AT END Imperative-Statement-2 ]

  [END-READ]
```

Format 2

```
READ File-Name-1 RECORD

  [ WITH [ { NO     } ] LOCK ]
           { KEPT   }
           { IGNORE }
           { WAIT   }

  [ ALLOWING UPDATERS ]

  [ INTO Identifier-1 ]

  [ KEY IS Record-Key-Name-1 ]

  [ INVALID KEY Imperative-Statement-3 ]

  [ NOT INVALID KEY Imperative-Statement-4 ]

  [END-READ]
```

Format 3

```
READ File-Name-1
  [ NEXT RECORD KEY IS DataName-1 ]
```

Syntax rules

All Formats

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 an alphanumeric group item or an elementary item of category alphanumeric or category national.
2. The storage area associated with identifier-1 and the record area associated with file-name-1 shall not be the same storage area.
3. 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.
4. WITH LOCK and WITH KEPT LOCK are synonymous.
5. WITH NO LOCK and ALLOWING UPDATERS are synonymous.

Format 1

6. PREVIOUS and BACKWARDS are synonymous.
7. None of the phrases 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.
8. 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.
9. 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: AT END, or NOT AT END.

10. If SIZE phrase is specified, identifier-2 shall be a numeric data item.

Format 2

11. The KEY phrase may be specified only if ORGANIZATION IS INDEXED is specified in the file control entry for file-name-1.
12. 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.
13. Data-name-1 or record-key-name-1 may be qualified.

Format 3

14. DataName-1 can be qualified, and references a data item whose declaration includes an IDENTIFIED BY clause and is included in the XD record declarations for File-Name-1.

General rules

All Formats

1. The open mode of the file connector referenced by file-name-1 shall be input or I-O. If it is any other value, the execution of the READ statement is unsuccessful and the I-O status value for file-name-1 is set based on your iscobol.file.status * setting.
2. The execution of the READ statement causes the value of the I-O status in the file connector referenced by file-name-1 to be updated.
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 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 same READ statement without the INTO phrase is executed.
   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 an alphanumeric 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 an alphanumeric group item of the maximum size established by the RECORD clause.
6. Whether record locking is in effect is determined by the LOCK MODE clause.
7. 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 that file connector, the record lock is ignored and the READ operation proceeds as if the record were not locked.
8. If 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 content of the associated record area is undefined.

D. The key of reference for indexed files is unchanged.

E. The READ statement is unsuccessful.

9. If record locks are in effect, the following actions take place:

   A. If single record locking is specified for the file connector associated with file-name-1, any prior record lock associated with that file connector is released by the 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 a successfully accessed record is set.

   D. If lock mode is manual, the record lock associated with a successfully accessed record is set only if the LOCK phrase is specified on the READ statement. The following types of lock are supported:

Type	Action
[KEPT] LOCK	a lock is acquired on the read record.
NO LOCK	a lock is not acquired on the read record, unless iscobol.file.index.read_lock_test (boolean) * is set to true in the configuration. In such case, a lock is acquired just to test the record status, then it's immediately released.
IGNORE LOCK	a lock is not acquired on the read record. The record is always read, regardless of it's locked or not.
WAIT LOCK	if a lock can't be acquired because the record is locked, the runtime waits for the record to be unlocked, then reads it with lock. This feature is fully supported by c-tree RTG. It is supported also by JIsam but, in a thin client or file server environment, it works only if iscobol.file.lock_manager * is set to "com.iscobol.as.locking.InternalLockManager".

10. 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:

       i. 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.

       ii. 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 fatal 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, if the NOT AT END phrase or NOT INVALID KEY phrase is specified, to 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.

11. If the number of bytes 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 bytes 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.
12. 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 runtime element 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.
13. Unless otherwise specified, at the completion of any unsuccessful execution of a READ statement, the content of the associated record area is undefined, the key of reference is undefined for indexed files, and the file position indicator is set to indicate that no valid record position has been established.

Format 1

14. 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.
15. 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, the I-O status value associated with file-name-1 is set based on your iscobol.file.status * setting, the at end condition exists.
    C. If the file position indicator was established by a prior OPEN or START statement, the first existing record that is selected is either:
        i. If NEXT is specified or implied, the first existing record in the physical file whose record number or key value is greater than or equal to the file position indicator, or
        ii. If PREVIOUS is specified, the first existing record in the physical 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.
    D. If the file position indicator was established by a prior READ statement and the file is an indexed file whose current key of reference does not allow duplicates or a sequential or relative file, the first existing record in the physical 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.

E. For indexed files, if the file position indicator was established by a prior READ statement, and the current key of reference does allow duplicates, the record that is selected is one of the following:

   i. If NEXT is specified or implied,

      a. If there exists in the physical file a record whose key value is equal to the file position indicator and whose logical position within the set of duplicates is after the record that was made available by that prior READ statement, the record within the set of duplicates that is immediately after the record that was made available by that prior READ statement;

      b. otherwise; the first record in the physical file whose key value is greater than the file position indicator.

   ii. If PREVIOUS is specified,

      a. If there exists in the physical file a record whose key value is equal to the file position indicator and whose logical position within the set of duplicates is before the record that was made available by that prior READ statement, the record within the set of duplicates that is immediately before the record that was made available by that prior READ statement;

      b. otherwise, the last record within the set of duplicates, if any, whose key value is the first key value 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 I-O status value associated with file-name-1 is set based on your iscobol.file.status * setting, the at end condition exists, the file position indicator is set to indicate that no next or previous logical record exists, and execution proceeds.

NOTE - Except in the case of an indexed file, the record made available may have a length of zero.

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, the I-O status value associated with file-name-1 is set based on your iscobol.file.status * setting, the at end condition exists, and execution proceeds.

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.

16. If the at end condition exists, the following occurs in the order specified:

   A. The I-O status value associated with file-name-1 is set based on your iscobol.file.status * setting to indicate the at end condition.

   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 exists, execution of the READ statement is unsuccessful.

17. For a relative file, if the RELATIVE KEY clause is specified for file-name-1, the execution of a 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.
18. 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.
19. The I-O status for the file connector referenced by file-name-1 is set based on your iscobol.file.status * setting if the execution of the READ statement is successful, an indexed file is being sequentially accessed, the key of reference is an alternate record key, and one of the following is true:

    A. the NEXT phrase is specified or implied and the alternate record key in the record that follows the record that was successfully read duplicates the same key in the record that was successfully read, or

    B. the PREVIOUS phrase is specified and the alternate record key in the record that immediately precedes the record that was successfully read duplicates the same key in the record that was successfully read.

20. If SIZE clause is used, identifier-2 is set, after a READ executed correctly, to the number of bytes read. It works on sequential files.

Format 2

21. 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.
22. 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 physical file does not contain such a record, the invalid key condition exists and execution of the READ statement is unsuccessful.
23. 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.
24. 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.
25. 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 in a sequence of duplicates that was released to the operating environment. 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.

Format 3

26. A Format 3 READ statement reads an XML document from an I/O stream or retrieves an XML element from the in-memory representation of an XML document.
27. The READ statement without NEXT KEY specified reads an XML document and creates an internal tree-based representation of that document. In addition, it populates the appropriate record definition from the XD with as much data as possible, based on the record elements and occurrence limitations of the associated XML tags. The populated elements are associated with their in-memory counterparts to allow DELETE, REWRITE, and WRITE to modify the internal representation.
28. READ NEXT finds the associated internal counterpart of the indicated key and either populates the key and all subordinate items with the next occurrence of that internal counterpart, or returns end-of-file if there is no next occurrence.
29. No stream I/O is actually performed by the READ NEXT KEY IS statement.

Examples

Format 1- Read next record sequentially validating when end of file is reached

```
read customers next
     at end display message "EOF has been reached"
            set customers-eof to true
     not at end display cust-code
end-read
```

Format 1 - Read previous record sequentially validating when top of file is reached

```
read customers previous
     at end display message "TOF has been reached"
            set customers-tof to true
     not at end display cust-code
end-read
```

Format 2 - Read record by primary key, checking for invalid key

```
move 1234 to cust-code
read customers
     invalid key display message "Customer Code : " cust-code " not found!"
     not invalid key display message "Successful Read"
end-read
```

Format 2 - Read record by alternate key, checking for invalid key

```
move "Smith" to cust-last-name
read customers key cust-last-name
     invalid key display message "Customer Last Name : " cust-last-name " not found!"
     not invalid key display message "Successful Read"
end-read
```

Read record and put the contents on different variable

```
read customers into ws-first-customer
read customers next into ws-last-customer
```

Read record with no lock

```
read customers with no lock
read customers next with no lock
```

RECEIVE

General Format

```
RECEIVE Dest-Item FROM { THREAD Thread-1    }
                       { LAST THREAD        }
                       { ANY { THREAD  }    }
                             { THREADS }

  [Remaining-Phrase]

  [ ON EXCEPTION Imperative-Statement-1 ]

  [ NOT ON EXCEPTION Imperative-Statement-2 ]

  [END-RECEIVE]
```

Remaining-Phrases are optional and can appear in any order.

```
   BEFORE TIME Timeout
   WITH NO WAIT

  THREAD IN Thread-2

  SIZE IN Size-Item

  STATUS IN Status-Item
```

Syntax rules

1. *Dest-Item* is any data item.
2. *Thread-1* and *Thread-2* are usage HANDLE or HANDLE OF THREAD data items.
3. *Timeout* is a numeric literal or data item.
4. *Size-Item* and, *Status-Item* are numeric data items.
5. *Size-Item*, *Status-Item* and *Thread-2* cannot be indexed or reference modified.
6. *Status-Item* is a two-character group item defined as, PIC X(02), or PIC 9(02).
7. *Imperative-Statement-1* and *Imperative-Statement-2* are any imperative statements.

General rules

1. The RECEIVE statement returns the next available message into dest-item. Messages are RECEIVEd as follows:
   A. FROM THREAD thread-1 specifies that only messages from the thread identified by thread-1 are allowed.
   B. FROM LAST THREAD specifies that only messages from the last thread are allowed
   C. FROM ANY THREAD specifies that all messages are allowed.

2. Messages are received in the order sent.
3. When BEFORE TIME is specified, the RECEIVE statement will time out after the specified number of hundredths of seconds. If this happens, the destination item (dest-item) is not updated. NO WAIT is equivalent to BEFORE TIME 0. If timeout is zero, then the RECEIVE statement times out immediately if a message is not available.
4. A timeout of zero is mandatory when receiving messages FROM ANY THREAD; other timeouts will not clean compile.
5. The thread ID of the sending thread is put into thread-2 by RECEIVE.
6. The size of the message sent is placed in size-item.
7. The status of the RECEIVE statement is placed in status-item as defined below:

"00"	Success - message received
"04"	Success - message received, but it was truncated
"10"	Exception - sending thread does not exist or terminated
"99"	Exception - timed out

8. If the RECEIVE statement is successful, statement-2 executes otherwise, statement-1 executes

Examples

Receive a message from a parent thread

```
receive msg-string from last thread
   before time 200
   not on exception
      if msg-string = "cancel thread"
         goback
      end-if
end-receive
```

RELEASE

General Format

```
RELEASE Record-name-1 [ FROM {Identifier-1} ]
                             {Literal-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 or national function. Record-name-1 and identifier-1 shall not reference the same storage area.
3. Identifier-1 or literal-1 shall be valid as a sending operand in a MOVE statement specifying record-name-1 as the receiving operand.

General rules

1. A RELEASE statement may 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 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

      or

      MOVE literal-1 TO record-name-1

      according to the rules specified for the MOVE statement.
   B. The same RELEASE statement without the FROM phrase.
4. If the number of bytes to be released to the sort operation is greater than the number of bytes in record-name-1, the content of the bytes that extend beyond the end of record-name-1 are undefined.

Examples

Use Release on input procedure of Sort to provide input records

```
input-output section.
file-control.
  select work-file assign to "sort-workfile".

file section.
sd work-file.
01 work-rec.
   03 wr-key-1 pic x(5).
   03 wr-key-2 pic x(5).

working-storage section.
01 eof-flag pic x.
   88 eof-sort value "Y" false "N".

procedure division.
main.
 display x"0d0a" "First sort descending on key 1, ascending on key 2"
 sort work-file on descending key wr-key-1
                   ascending  key wr-key-2
      input procedure is input-proc
      output procedure is output-proc.
 display x"0d0a" "Second sort descending on key 2, ascending on key 1"
 sort work-file on descending key wr-key-2
                   ascending  key wr-key-1
      input procedure is input-proc
      output procedure is output-proc.
 goback.

 input-proc.
    release work-rec from "aaaaabbbb1"
    release work-rec from "aaaaazzzz2"
    release work-rec from "ccccczzzz3"
    release work-rec from "cccccdddd4"
    release work-rec from "cccccmmmm5"
    release work-rec from "zzzzzcccc6"
    release work-rec from "zzzzzaaaa7"
    release work-rec from "zzzzznnnn8".

 output-proc.
     set eof-sort to false
     perform until eof-sort
       return work-file
        at end set eof-sort to true
        not at end display work-rec
       end-return
     end-perform.
```

RESUME

General Format

```
RESUME Procedure-Name
```

Syntax rules

1. Procedure-Name is a nonnumeric literal or an alphanumeric data item.

General rules

1. A declarative procedure is said to complete normally unless a RESUME statement occurs.

Examples

```
DECLARATIVES.
SectionName  SECTION.
    USE  AFTER  STANDARD  ERROR  PROCEDURE  ON  FileName.
    RESUME  ParagraphName
END  DECLARATIVES.
```

RETURN

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.
4. If identifier-1 is a strongly-typed group item, there shall be exactly one record area subordinate to the SD for file-name-1. This record area shall be a strongly-typed group item of the same type as identifier-1.

General rules

1. A RETURN statement may 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 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 is set to exist 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 exists, execution of the RETURN statement is unsuccessful and the contents of the record area associated with file-name-1 are undefined.
4. If an at end condition does not exist 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 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 same RETURN statement without the INTO phrase is executed.
   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 implied MOVE statement does not occur if the execution of the RETURN statement was unsuccessful. 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.

Examples

Use Return on output procedure of Sort to provide output records

```
input-output section.
file-control.
  select work-file assign to "sort-workfile".

file section.
sd work-file.
01 work-rec.
   03 wr-key-1 pic x(5).
   03 wr-key-2 pic x(5).

working-storage section.
01 eof-flag pic x.
   88 eof-sort value "Y" false "N".

procedure division.
main.
 display x"0d0a" "First sort descending on key 1, ascending on key 2"
 sort work-file on descending key wr-key-1
                   ascending  key wr-key-2
      input procedure is input-proc
      output procedure is output-proc.
 display x"0d0a" "Second sort descending on key 2, ascending on key 1"
 sort work-file on descending key wr-key-2
                   ascending  key wr-key-1
      input procedure is input-proc
      output procedure is output-proc.
 goback.

 input-proc.
    release work-rec from "aaaaabbbb1"
    release work-rec from "aaaaazzzz2"
    release work-rec from "ccccczzzz3"
    release work-rec from "cccccdddd4"
    release work-rec from "cccccmmmm5"
    release work-rec from "zzzzzcccc6"
    release work-rec from "zzzzzaaaa7"
    release work-rec from "zzzzznnnn8".

 output-proc.
     set eof-sort to false
     perform until eof-sort
       return work-file
        at end set eof-sort to true
        not at end display work-rec
       end-return
     end-perform.
```

REWRITE

Format 1

```
REWRITE Record-Name-1 RECORD [ FROM Identifier-1 ]

  [ WITH [NO] LOCK ]

  [ INVALID KEY Imperative-Statement-1 ]

  [ NOT INVALID KEY Imperative-Statement-2 ]

  [END-REWRITE]
```

Format 2

```
REWRITE Record-Name-1

  [ KEY IS DataName-1 ]
```

Syntax Rules

1. 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. Neither the INVALID KEY phrase nor the NOT INVALID KEY phrase shall be specified for a REWRITE statement that references a file with sequential organization or a file with relative organization and sequential access mode.
4. If automatic locking has been specified for the rewrite file, neither the WITH LOCK phrase nor the WITH NO LOCK phrase shall be specified.
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. Data-name-1 can be qualified, and references a data item whose declaration includes an IDENTIFIED BY clause and is included in the XD record declarations for file-name-1.

General Rules

Format 1

1. The rewrite file connector is the file connector referenced by the file-name associated with record-name-1.
2. The rewrite file connector shall have an open mode of I-O. If the open mode is some other value or the file is not open, the I-O status in the rewrite file connector is set based on your iscobol.file.status * setting and the execution of the REWRITE statement is unsuccessful.
3. The successful execution of the REWRITE statement releases a logical record to the operating environment.
4. If the rewrite file connector has an access mode of sequential, the immediately previous input-output statement executed that referenced this file connector shall have been a successfully executed READ statement. If this is not true, the I-O status in the rewrite file connector is set based on your iscobol.file.status * setting and the execution of the REWRITE statement is unsuccessful. For a successful REWRITE statement, the operating environment logically replaces the record that was accessed by the READ statement.

NOTE - Logical records in relative and sequential files may have a length of zero. Logical records in an indexed file shall always be long enough to contain the record keys.

5. 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.
6. The figurative constant SPACE when specified in the REWRITE statement references one alphanumeric space character.
7. If record locks are in effect, the following actions take place at the beginning or at the successful completion of the execution of the REWRITE statement:
   A. If single record locking is specified for the rewrite file connector:
      i. If that file connector holds a record lock on the record to be logically replaced, that lock is released at completion unless the WITH LOCK phrase is specified.
      ii. If that file connector holds a record lock on a record other than the one to be logically replaced, that lock is released at the beginning.
   B. If multiple record locking is specified for the rewrite file connector, and a record lock is associated with the record to be logically replaced, that record lock is released at completion only when the WITH NO LOCK phrase is specified and the record to be logically replaced was already locked by that file connector.
   C. If the WITH LOCK phrase is specified, the record lock associated with the record to be replaced is set at completion.
8. The file position indicator in the rewrite file connector is not affected by the execution of a REWRITE statement.
9. The execution of the REWRITE statement causes the I-O status value in the rewrite file connector to be updated.
10. If the execution of the REWRITE statement is unsuccessful, no logical record updating takes place, the content of the record area is unaffected, and the I-O status in the rewrite file connector is updated as indicated in other general rules.
11. When record-name-1 is specified, if the number of bytes to be written to the file is greater than the number of bytes in record-name-1, the content of the bytes that extend beyond the end of record-name-1 are undefined.

Sequential Files

12. If the number of bytes in the data item referenced by identifier-1, the runtime representation of literal-1, or the record referenced by record-name-1 is not equal to the number of bytes in the record being replaced, the execution of the REWRITE statement is unsuccessful and the I-O status in the rewrite file connector is set based on your iscobol.file.status * setting.

Relative and Indexed Files

13. The number of bytes in the record referenced by identifier-1, the runtime representation of literal-1, or the record referenced by record-name-1 may differ from the number of bytes in the record being replaced.

14. 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.
15. The number of bytes in the runtime representation of literal-1, the data item referenced by identifier-1, or the record referenced by record-name-1 after any changes made to the record length by the FORMAT clause shall not be larger than the largest or smaller than the smallest number of bytes allowed by the RECORD IS VARYING clause associated with file-name-1 or the file-name associated with record-name-1. If this rule is violated, the execution of the REWRITE statement is unsuccessful and the I-O status in the rewrite file connector is set based on your iscobol.file.status * setting.

Relative Files

16. For a file accessed in either random or dynamic access mode, the operating environment 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 and the I-O status in the rewrite file connector is set to the invalid key condition '23'.

Indexed Files

17. If the access mode of the rewrite file connector is random or dynamic, the record to be replaced is specified by the prime record key. If there is no existing record in the physical file with that prime record key, the execution of the REWRITE statement is unsuccessful and the I-O status in the rewrite file connector is set to the invalid key condition, '23'.
18. 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 where the alternate record key value is equal to the same alternate key value in one or more records in the file based on the collating sequence for the file.
19. The comparison for equality for record keys is based on the collating sequence for the file according to the rules for a relation condition. The invalid key condition exists under the following circumstances:
    A. When the rewrite file connector 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 that physical file, the I-O status associated with the rewrite file connector is set to '23'.
    B. When an alternate record key of the record to be replaced does not allow duplicates and the value of that alternate record key is equal to the value of the corresponding alternate record key of a record in that physical file, the I-O status associated with the rewrite file connector is set to '22'.

       When the invalid key condition is recognized, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, and the content of the record area is unaffected.

Format 2

20. A Format 2 REWRITE statement modifies an XML element in the in-memory representation of the XML document.
21. No stream I/O is performed by the REWRITE verb.
22. REWRITE KEY IS updates the in-memory representation of the elements and child elements associated with data-name-1. All other elements in the in-memory representation remain unchanged.
23. If the internal representation of the XML document is modified through the use of WRITE KEY, REWRITE KEY,

or DELETE KEY, and the internal representation is cleared with either a CLOSE or READ (no key) statement, then the CLOSE or READ statement returns a status of -10, indicating the operation succeeded but no write was done.

Examples

Update last read record validating the key

```
move 1234 to cust-code
read customers
add ws-invoice-amount to cust-total-debt
rewrite customer-record
 invalid key display message "Customer could not be updated. Code : " cust-code
 not invalid key display message "Customer updated. Code : " cust-code
end-rewrite
```

ROLLBACK

General Format

```
ROLLBACK TRANSACTION
```

General rules

Each system's native mechanism for transaction management is invoked.

1. When ROLLBACK is enabled in the FILE-CONTROL entry for a file, the record and file locking rules are extended for that file. ROLLBACK verb removes these locks.
2. If an Indexed or Relative data file is CLOSEd during a transaction the CLOSE will wait for the completed ROLLBACK.
3. If an abnormal end of program occurs, an automatic rollback is initiated.

Examples

```
*> This sample will work only with a file system that supports transactions
*> The select should include lock clause with rollback: lock automatic with rollback
...
input-output section.
file-control.
select invoices assign to inv-path
   organization indexed
   access dynamic
   record key cust-code
   lock mode manual with rollback |the rollback clause is required to
   status inv-status.             |activate transaction management
...
procedure division.
...
   start transaction
   display "Customer to apply 10% discount? "
   accept ws-cust-code
   move ws-cust-code to cust-code
   read customers
        invalid key display message "Customer not found"
                    rollback
                    exit paragraph
   end-read
   accept ws-date from date
   move ws-date to cust-last-date-discount
   rewrite cust-rec
   move ws-cust-code to inv-cust-code
   start invoices key = inv-cust-code
         invalid key set end-of-invoices to true
         not invalid key set end-of-invoices to false
   end-start
   perform until end-of-invoices
     read invoices next at end exit perform
          not at end
              if inv-cust-code not = ws-cust-code
                 exit perform
              end-if
     end-read
     if inv-paid-status = "N"
        move 10 to inv-discount
        rewrite invoice-rec
     end-if
   end-perform
   display "Changes applied, confirm Commit: (Y/N)"
   accept apply-commit
   if apply-commit = "Y"
      commit *> All changes are definitely applied
   else
      rollback *> All changes get undone
   end-if.
```

SEARCH

Format 1

```
SEARCH Identifier-1 [ VARYING {Identifier-2} ]
                              {Index-Name-1}

  [ AT END Imperative-Statement-1 ]

  { WHEN Condition-1 { Imperative-statement-2 } } ...
                     { NEXT SENTENCE          }

  [END-SEARCH]
```

Format 2

```
SEARCH ALL Identifier-1

  [ AT END Imperative-Statement-1 ]

  WHEN {Data-Name-1 {IS EQUAL TO} {Identifier-3           } }
                    {IS =       } {Literal-1              }
                                  {Arithmetic-Expression-1}

  [AND {Data-Name-2 {IS EQUAL TO} {Identifier-4           } } ] ...
                    {IS =       } {Literal-2              }
                                  {Arithmetic-Expression-2}

  { Imperative-statement-2 } }
  { NEXT SENTENCE          }

  [END-SEARCH]
```

Syntax rules

All Formats

1. Identifier-1 shall not be subscripted or reference modified and its data description entry shall contain an OCCURS clause including an INDEXED phrase.

Format 1

2. Identifier-2 shall reference a data item whose usage is index or a data item that is an integer. Identifier-2 shall not be subscripted by the first or only index-name specified in the INDEXED phrase in the OCCURS clause specified in the data description entry for identifier-1.
3. Condition-1 may be any conditional expression.

Format 2

4. The OCCURS clause associated with identifier-1 shall contain the KEY phrase.
5. Data-name-1 and all repetitions of data-name-2 may be qualified.
6. All referenced condition-names shall be defined as having only a single value and shall be subscripted by the first index-name associated with identifier-1, along with any subscripts required to uniquely identify the condition-name. The data-name associated with each condition-name shall be specified in the KEY phrase in the OCCURS clause associated with identifier-1. The index-name subscript shall not be followed by a '+' or a '–

'.

7. Identifier-3, identifier-4, identifiers specified in arithmetic-expression-1, and identifiers specified in arithmetic-expression-2 shall be neither referenced in the KEY phrase of the OCCURS clause associated with identifier-1 nor subscripted by the first index-name associated with identifier-1.
8. When a data-name in the KEY phrase in the OCCURS clause associated with identifier-1 is referenced or when a condition-name associated with a data-name in the KEY phrase in the OCCURS clause associated with identifier-1 is referenced, all preceding data-names in that KEY phrase or their associated condition-names shall also be referenced.

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 is 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:

      i. 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.
      ii. If the AT END phrase is not specified and the exception condition was raised during the execution of the SEARCH statement, control is transferred to the end of the SEARCH statement.
      iii. If the AT END phrase is not specified and neither exception condition was raised because the checking for those exception conditions was not enabled, control is transferred to the end of the SEARCH statement.
2. The comparison associated with each WHEN phrase is executed in accordance with the rules specified for conditional expressions.

Format 1

3. 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 3a and the following also applies:

   i. 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.

   ii. 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:

   i. 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.

   ii. 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 3a. 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.

4. 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. If, at the start of the execution, the search index contains a value that corresponds to an occurrence number that is negative, zero, or greater than the highest permissible occurrence number for identifier-1, the search operation is unsuccessful, and execution proceeds as indicated in general rule 1b. 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, in which case the search operation is unsuccessful and execution proceeds as indicated in general rule 1b. 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

5. At the start of the execution of a SEARCH statement with the ALL phrase specified, the following conditions shall be true:

   A. 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.

   B. 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.

6. If any condition specified in general rule 5 is not satisfied:

   A. If one or more settings of the search index satisfy all conditions in the WHEN phrase, one of the following occurs:

      i. the final setting of the search index is set equal to one of those settings, but it is undefined which one; execution proceeds as in general rule 1a;

ii. the final setting of the search index is undefined and execution proceeds as in general rule 1b.

It is undefined which of these alternatives occurs.

B. If no such setting of the search index exists, the final setting of the search index is undefined, execution proceeds as in general rule 1b.

7. If both conditions specified in general rule 5 are satisfied and there is more than one setting of the search index for which all conditions in the WHEN phrase can be satisfied, the search operation is successful. The final setting of the search index is equal to one of them, but it is undefined which one.
8. 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.

Examples

Format 1 - Search by name on a staff table

```
working-storage section.
01 staff-table occurs 5 times indexed by table-idx.
   05 tbl-code pic 9(2).
   05 tbl-name pic x(10).
   05 tbl-salary pic 9(6).
77 search-name pic x(10).

procedure division.
main.
  perform fill-table
  perform search-table
  goback.

fill-table.
  move 1         to tbl-code(1)
  move "Adam"    to tbl-name(1)
  move 84000     to tbl-salary(1)
  move 2         to tbl-code(2)
  move "Eve"     to tbl-name(2)
  move 74000     to tbl-salary(2)
  move 3         to tbl-code(3)
  move "Jack"    to tbl-name(3)
  move 93500     to tbl-salary(3)
  move 4         to tbl-code(4)
  move "Allan"   to tbl-name(4)
  move 63400     to tbl-salary(4)
  move 5         to tbl-code(5)
  move "Lilian"  to tbl-name(5)
  move 53300     to tbl-salary(5)
  .

search-table.
  move "Jack" to search-name
  move 1 to table-idx
  search staff-table
    at end display message "Name not found"
    when tbl-name(table-idx) = search-name
           display message
             tbl-code(table-idx) ", " tbl-name(table-idx) ", " tbl-salary(table-idx)
  end-search.
```

Format 2 - Search all by name on staff table

```
working-storage section.
77 search-name pic x(10).
01 staff-table occurs 5 times
               ascending key is tbl-first-name
               indexed by table-idx.
   05 tbl-code       pic 9(2).
   05 tbl-first-name pic x(10).
   05 tbl-last-name  pic x(10).
   05 tbl-salary     pic 9(6).

procedure division.
main.
  perform fill-table
  perform search-all-table
  goback.

fill-table.
  move 1         to tbl-code(1)
  move "Adam"    to tbl-first-name(1)
  move "Smith"   to tbl-last-name(1)
  move 84000     to tbl-salary(1)

  move 2         to tbl-code(2)
  move "Eve"     to tbl-first-name(2)
  move "Green"   to tbl-last-name(2)
  move 74000     to tbl-salary(2)

  move 3         to tbl-code(3)
  move "Jack"    to tbl-first-name(3)
  move "Yellow"  to tbl-last-name(3)
  move 93500     to tbl-salary(3)

  move 4         to tbl-code(4)
  move "Allan"   to tbl-first-name(4)
  move "Poe"     to tbl-last-name(4)
  move 63400     to tbl-salary(4)

  move 5         to tbl-code(5)
  move "Jack"    to tbl-first-name(5)
  move "Samco"   to tbl-last-name(5)
  move 53300     to tbl-salary(5)
  .

search-all-table.
  display "----- Search All"
  move "Jack" to search-name
  move 1 to table-idx

  search all staff-table
    at end display "Name not found"
    when tbl-first-name(table-idx) = search-name
           display
             tbl-code(table-idx) ", " tbl-first-name(table-idx) ", "
             tbl-last-name(table-idx) ", " tbl-salary(table-idx)
  end-search.
```

SEND

General Format

```
SEND { Literal-1   } TO { THREAD Dest-Thread }
     { Data-Item-1 }    { LAST THREAD        }
                        { ALL {THREAD }      }
                              {THREADS}
```

Syntax rules

1. *Literal-1* is a literal or data item.
2. *Dest-Thread* is a USAGE HANDLE or HANDLE OF THREAD data item.

General rules

1. The SEND statement sends a message containing the data in Literal-1 to one or more threads using the following logic:
   A. THREAD Dest-Thread causes the message to be sent to the thread identified by Dest-Thread. More than one Dest-Thread can be specified.
   B. LAST THREAD causes the message to be sent to the last thread
   C. ALL THREADS causes the message to be sent to all currently existing threads, except the sending thread.
2. The size of the message is equal to the size of Literal-1.
3. The THREAD Dest-Thread and LAST THREAD options create a queued message for the specified Dest-THREAD.
4. The ALL THREADS option creates a broadcast message which are available by any Thread.

Examples

Send a message to the parent thread

```
send "cancel main" to last thread
```

SERVICE RELOAD

General Format

```
SERVICE RELOAD identifier-1
```

In OS/VS COBOL, for programs to be executed under CICS, the SERVICE RELOAD statement is required to ensure addressability of items defined in the LINKAGE SECTION.

The SERVICE RELOAD statement is supported for compatibility with OS/VS COBOL and is treated as a comment.

SET

Format 1

```
SET { index-name-1 } ... { TO } { index-name-2           }
    { identifier-1 }     { =  } { identifier-2           }
```

Format 2

```
SET { index-name-3 } ... { UP   } BY identifier-2
                         { DOWN }
```

Format 3

```
SET { { mnemonic-name-1 } ... { TO } { ON  } } ...
                              { =  } { OFF }
```

Format 4

```
SET { { condition-name } ... { TO } { TRUE  } } ...
                             { =  } { FALSE }
```

Format 5

```
SET FILE-PREFIX { TO } File-Prefix
                { =  }
```

Format 6

```
SET { CONFIGURATION } { Env-Name { TO } Env-Value } ...
    { ENVIRONMENT   }            { =  }

  [ ON EXCEPTION Statement-1 ]

  [ NOT ON EXCEPTION Statement-2 ]

[END-SET]
```

Format 7

```
SET Identifier-5 { TO } { { ADDRESS } OF Identifier-6 }
                 { =  }   { HANDLE  }
                        { NULL                        }
```

Format 8

```
SET Result-Item { TO } SIZE OF Data-Item
                { =  }
```

Format 9

```
SET { ADDRESS } OF  Linkage-Item { TO } { Pointer              }
    { HANDLE  }                  { =  } { ADDRESS OF Data-Item }
```

Format 10

```
SET { INPUT        } WINDOW { TO } Window-1
    { INPUT-OUTPUT }        { =  }
    { I-O          }
    { OUTPUT       }
```

Format 11

```
SET { Handle-1 } ... { TO } HANDLE OF { Screen-1        }
                     { =  }           { CONTROL ID id-1 }
```

Format 12

```
SET THREAD Thread-id PRIORITY { TO } Priority
                              { =  }
```

Format 13

```
SET EXCEPTION { VALUE  }  Exc-Value { TO } { CUT-SELECTION        }  ...
              { VALUES }            { =  } { COPY-SELECTION       }
                                           { PASTE-SELECTION     }
                                           { DELETE-SELECTION    }
                                           { UNDO                }
                                           { REDO                }
                                           { SELECT-ALL-SELECTION }
                                           { ITEM-HELP           }
```

Format 14

```
SET Result-Item { TO } { Class:>{ method  ( [parameters] ) } }
                { =  }          { field                    }
                       { ObjRef:>{ method  ( [parameters] ) } }
                                 { field                    }
                       {   SELF:>{ method  ( [parameters] ) } }
                                 { field                    }
                       {  SUPER:>{ method  ( [parameters] ) } }
                                 { field                    }
                       { SELF                                 }
```

Format 15

```
SET Result-Item { TO } CAPACITY OF Dynamic-Capacity-Table
                { =  }
```

Syntax rules

Format 1

1. Identifier-1 shall reference a data item of class index or an integer data item.
2. Identifier-2 shall reference a data item of class index.
3. If identifier-1 references a numeric data item, index-name-2 shall be specified.
4. Result-Item shall reference a data item
5. Class shall reference a class described in the REPOSITORY paragraph.
6. ObjRef shall reference a data item of type OBJECT REFERENCE Class.

Format 3

7. Mnemonic-name-1 shall be associated with an external switch, the status of which may be altered.

Format 4

8. Condition-name-1 shall be associated with a conditional variable.
9. 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 5

10. File-prefix references a nonnumeric literal or alphanumeric data item.

Format 6

11. Env-name references a nonnumeric literal or data item.
12. Env-value references a USAGE DISPLAY numeric or nonnumeric literal or data item. If numeric, it must be an integer
13. CONFIGURATION and ENVIRONMENT are synonymous.

Format 7

14. Identifier-5 shall reference a data item of category data-pointer. Identifier-6 shall be of category data-pointer.
15. Data-name-1 shall be a based data item.
16. If identifier-5 references a restricted data-pointer, identifier-6 shall be the predefined address NULL or shall reference a data-pointer restricted to the same type. If data-name-1 is a strongly-typed group item or a restricted pointer, identifier-6 shall reference a data-pointer restricted to the type of data-name-1.

    If identifier-6 references a restricted data-pointer, either identifier-5 shall reference a data-pointer restricted to the same type or data-name-1 shall be a typed item of the type to which identifier-6 is restricted.

Format 8

17. Result-item must be a numeric data item

Format 9

18. Linkage-item must be declared in the Linkage section.
19. Pointer must be a data item with USAGE POINTER.

Format 10

20. Window-1 is a USAGE HANDLE or PIC X(10) data item that refers to a floating window or the main application window

Format 11

21. Handle-1 is a USAGE HANDLE data item.
22. Screen-1 must refer to an elementary Screen Section item that describes a graphical control.
23. Id-1 references a numeric literal or data item.

Format 12

24. Thread-id references a USAGE HANDLE or HANDLE OF THREAD data item.
25. Priority references a numeric literal or data item.

Format 13

26. Exc-value references an integer literal or data item.
27. `VALUE` and `VALUES` are synonymous.

Format 14

28. Class is a class defined in the REPOSITORY paragraph

29. ObjRef is a data item whose usage is OBJECT REFERENCE
30. method and field are the exact name of a method or a field exposed by the invoked class

parameters can be any kind of item depending on the invoked method signature

Format 15

31. Result-item must be a numeric data item
32. Dynamic-Capacity-Table is a data item for which the OCCURS DYNAMIC clause is specified

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.

Format 1

2. The following occurs for each recurrence of index-name-1 or identifier-1. Each time, the value of the sending operand is used as it was at the beginning of the execution of the statement. Item identification of the data item referenced by identifier-1 is done immediately before the value of that data item is changed:
    A. If index-name-1 is specified:
        i. If identifier-2 is specified, the content of the data item referenced by identifier-2 is placed in the index referenced by index-name-1 unchanged.
        ii. If index-name-2 is specified:
            a. 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. Otherwise, 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 associated with index-name-2.
    C. If identifier-1 references an index data item, the content of index-name-2 or of the data item referenced by identifier-2 is placed in the index referenced by index-name-1 unchanged.
    D. If identifier-1 references a numeric data item, that item is set to the occurrence number of the table element referenced by index-name-2.

Format 2

3. The following occurs for each recurrence of index-name-3. Each time, the value of identifier-2 is used as it was at the beginning of the execution of the statement:
    A. If identifier-2 does not evaluate to an integer, the execution of the SET statement is unsuccessful, and the content of the receiving operand is unchanged.
    B. If the occurrence number represented by the content of index-name-3 incremented (UP BY) or decremented (DOWN BY) the result of the evaluation of identifier-2 is a valid subscript for the table associated with index-name-3, index-name-3 is set to a value causing it to refer to the table element that corresponds in occurrence number to that subscript; otherwise, the execution of the SET statement is unsuccessful, and the content of the receiving operand is unchanged.

Format 3

4. 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.

Format 4

5. If the TRUE phrase is specified, the literal in the VALUE clause associated with condition-name-1 is placed in

the conditional variable according to the rules for the VALUE clause. If the length of the conditional variable is zero, the SET statement leaves it unchanged. 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.

6. 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. If the length of the conditional variable is zero, the SET statement leaves it unchanged.
7. If multiple condition-names are specified, the results are the same as if a separate SET statement had been written for each condition-name-1 in the same order as specified in the SET statement.

Format 5

8. FILE_PREFIX is an isCOBOL Framework Property used to list the paths in which to search for data files. Paths must be separated by the "\n" character sequence or by the current operating system path separator. The prefix "iscobol" will be automatically added by the Runtime. FILE_PREFIX is supported for backward compatibility.
9. A Format 5 SET statement is equivalent to this Format 6 SET statement:

```
SET ENVIRONMENT "FILE-PREFIX" TO file-prefix
```

Format 6

10. The Runtime Framework can be customized with the use of properties. Most of them are initially set to a default value. If a property does not have a default value or is not set, it is ignored. If the property is boolean you can use a value of '1', 'true', 'yes', or 'on' to set it to true. Any other value will set it to false. Refer to Configuration for general information about setting runtime properties.
11. The Format 6 SET statement can be used to modify these values at Runtime.
12. Env-name is the name of the Runtime Framework property variable to set. In it, lower-case characters are treated as upper case, and underscores are treated as hyphens. The first space character delimits the name. Env-name may specify either the literal name of the variable or a data-item whose value is the name of the variable. If you specify the actual name of the variable, then it must be enclosed the name in quotes. Env-value is the value to set the Framework property to.If it is a numeric data item, then the numeric value is stored, unless the program was compiled with -cdlz option, in such case the memory content is stored.
13. If the Env-Name is the name of a configuration variable whose value cannot be changed, then Statement-1 (if specified) is executed. Otherwise, Statement-2 (if specified) is executed.

Format 7

14. If identifier-5 is specified, the address identified by identifier-6 is stored in each data item referenced by identifier-5 in the order specified. Item identification of the data item referenced by identifier-5 is done immediately before the value of that data item is changed
15. If data-name-1 is specified, the address identified by identifier-6 is assigned to each based item referenced by data-name-1 in the order specified.
16. If the NULL option is used, then identifier-5 is set to point to no data item.

Format 8

17. The number of standard character positions occupied by data-item is stored in result-item

Format 9

18. If pointer is specified, then the address of the linkage-item is set to pointer. If the ADDRESS OF option is used, then the address of the linkage-item is set to the address of data-item. If the NULL option is used, then the address of the linkage-item is set to point to no data item.
19. The linkage-item must be defined as either level 01 or 77.
20. If the linkage-item is not listed in the PROCEDURE DIVISION USING phrase and is referenced before the SET

ADDRESS OF statement, then the runtime will abort with the exception message, "java.lang.NullPointer.Exception caught ! (null) ".

Format 10

21. Format 10 of the SET verb makes window-1 the current, or current and active window. The current window is the window to which DISPLAY statements refer. The active window is the window that is highlighted and the one to which user input is directed. Window-1 must be a handle to a valid floating window. If window-1 does not refer to a valid floating window, the SET statement has no effect.
22. INPUT, INPUT-OUTPUT, and I-O are synonymous. They cause window-1 to become both the current and active window.
23. OUTPUT causes window-1 to become the current window.

Format 11

24. A Format 11 SET statement retrieves the handle to the control described by screen-1 or id-1 and stores it in handle-1. Id-1 must have a value greater than zero. If a matching control is found, handle-1 is set to the handle of that control. If no matching control is found, handle-1 is set to NULL. If more than one control has a matching ID, then handle-1 is arbitrarily set to one of those controls

Format 12

25. A Format 12 SET statement sets the execution priority of a thread.
26. If thread-id is specified and identifies an existing thread, then the priority being set is for this thread. Otherwise, the current thread's priority is set to priority. If thread-id does not correspond to an existing thread, then the SET statement has no effect.
27. The thread priority can have these values:
    A. The minimum priority for a thread is "1".
    B. The maximum priority for a thread is "10".
    C. The normal (and default) priority for a thread is "5".

Format 13

28. A Format 13 SET statement associates the exception value specified in exc-value with an automated action that the Runtime can perform. Any keystroke, menu item, or control that produces the exc-value exception value will automatically cause the associated action to be performed.

The following actions can be used in association with the current control, if it is an entry-field; otherwise, they have no effect.

CUT-SELECTION	Cuts the current selection to the clipboard
COPY-SELECTION	Copies the current selection to the clipboard
PASTE-SELECTION	Pastes the clipboard into the entry field at the current location (replaces any existing selection)
DELETE-SELECTION	Deletes the current selection
UNDO	Undoes the last change
REDO	Redoes the change.
SELECT-ALL-SELECTION	Selects all the text in the entry field. In a multi-line entry field, this includes the text in all lines.

The ITEM-HELP action produces context-sensitive help for the control with the current input focus. See Help automation for more details.

Format 14

29. A Format 14 SET statement returns the result of a method invokation or the value of a field in object oriented programming.
30. Result-Item should be defined according to the result of the method and the field type. To receive strings and numbers, Result-Item can be a standard COBOL data-item with picture PIC X(n) or PIC 9(n). To receive an instance of an object, then Result-Item should be defined as OBJECT REFERENCE to that object.
31. If the field is a COBOL data item, it must be indicated upper case with hyhpens replaced by underscores, that is the way isCOBOL internally defines data items. For example, having `77 my-item pic x`, you will reference it using `SELF:>MY_ITEM` and not `SELF:>my-item`.
32. SELF means the current class. It can be used in a OBJECT paragraph. In a FACTORY paragraph the class logical name must be used in order to reference the current class.

```
IDENTIFICATION DIVISION.
CLASS-ID. MY_OBJ AS "MyObj".

IDENTIFICATION DIVISION.
OBJECT.

DATA DIVISION.
WORKING-STORAGE SECTION.
PROTECTED.
77 Item-1 PIC X(3).

PROCEDURE DIVISION.
IDENTIFICATION DIVISION.
METHOD-ID. TEST_SELF AS "TestSelf".
PROCEDURE DIVISION.
MAIN.
    SELF:>method1().
    DISPLAY SELF:>ITEM_1.
END METHOD.

IDENTIFICATION DIVISION.
METHOD-ID. METHOD1 as "method1".
PROCEDURE DIVISION.
MAIN.
    CONTINUE.
END METHOD.

END OBJECT.
```

```
IDENTIFICATION DIVISION.
CLASS-ID. MY_CLASS AS "MyClass".

IDENTIFICATION DIVISION.
FACTORY.

DATA DIVISION.
WORKING-STORAGE SECTION.
PROTECTED.
77 Item-1 PIC X(3).

PROCEDURE DIVISION.
IDENTIFICATION DIVISION.
METHOD-ID. TEST_SELF AS "TestSelf".
PROCEDURE DIVISION.
MAIN.
    MY_CLASS:>method1().
    DISPLAY MY_CLASS:>ITEM_1.
END METHOD.

IDENTIFICATION DIVISION.
METHOD-ID. METHOD1 as "method1".
PROCEDURE DIVISION.
MAIN.
    CONTINUE.
END METHOD.

END FACTORY.
```

33. SUPER refers to the class that the current class is inheriting from, if any.
34. Only PROTECTED fields can be referenced from FACTORY paragraphs and superclasses.
35. Setting Result-Item to SELF allows you to retrieve the instance of the current class.

Format 15

36. A Format 15 SET statement allows you to retrieve the exact number of items currently stored in a dynamic capacity table. This statement is particularly useful when dynamic capacity tables are nested and their CAPACITY data item is set to the highest capacity between the sibling dynamic capacity tables.

Examples

Format 1 and 2 - Set index name to specific value or increase current value

```
set idx-1 to 2
set idx-1 up by 4
```

Format 3 - Set a switch on

```
special-names.
  switch-0 is my-switch-a
  switch-1 is my-switch-b.
...
procedure division.
main.
   set my-switch-a to on.
   set my-switch-b to off.
```

Format 4 - Set conidition name to true/false

```
working-storage section.
01 customers-eof pic x.
   88 is-eof   value "Y" false "N".
01 applied-discount pic x.
   88 is-discounted value "Y" false "N".
...
procedure division.
main.
  set is-eof to false.
  set is-discounted to true.
```

Format 5 - Set the isCOBOL file.prefix runtime property

```
set file-prefix to "c:/mydir1/data1;c:/mydir2/data2"
```

Format 6 - Set a couple of isCOBOL runtime properties

```
set environment "file.prefix" to "c:/mydir1/data1;c:/mydir2/data2"
set environment "array_check" to "1"
```

Format 7 - Get the pointer to the area of an alphanumeric variable (requires -cp compiler option)

```
set my-pointer to address of employee-name
```

Format 8 - Get the size of some items

```
working-storage section.
77 num-comp   pic s9(7) comp-5.
77 str-var    pic x any length.
77 item-size  pic 9(4).
...
procedure division.
main.
  set item-size to size of num-comp
  *> item-size value will be : 4
  move "hello world" to str-var
  set item-size to size of str-var
  *> item-size value will be : 11
```

Format 9 - Set address of linkage item to address of working-storage item

```
working-storage section.
77 ws-par-1   pic x(10) value "no param".

linkage section.
01 par-1  pic x(10).
...
procedure division using par-1.
main.
   set address of par-1 to address of ws-par-1
   display par-1. *> Displays : no param
```

Format 10 - Activate different windows to display at them

```
working-storage section.
77 win-1  usage handle of window.
77 win-2  usage handle of window.
77 win-3  usage handle of window.
...
procedure division.
main.
   display standard window
           screen line 5 screen col 5
           handle win-1
   display independent window
           screen line 150 screen col 500
           handle win-2
   display independent window
           screen line 350 screen col 700
           handle win-3

   set i-o window win-2
   display "Message to Window 2" at line 10 col 20

   set i-o window win-3
   display "Message to Window 3" at line 10 col 20

   set i-o window win-1
   display "Message to Window 1" at line 10 col 20

   display message "done".
```

Format 11 - Sets a generic Entry-Field handle to the handle of the first field in the Screen

```
       working-storage section.
       77  generic-ef handle of entry-field.

       screen section.
       01  screen1.
           03 screen1-ef1 entry-field line 2 col 2.
           03 screen1-ef2 entry-field line 4 col 2.

       procedure division.
       main.
           display standard graphical window.
           display screen1.
           set generic-ef to handle of screen1-ef1.
```

Format 12 - Create a thread with the lowest priority

```
       working-storage section.
       77  th-1 handle of thread.

       procedure division.
       main.
           call thread "procedure1" handle th-1.
           set thread th-1 priority to 1.
```

Format 13 - Associate the key status value 1001 to the function CUT-SELECTION

```
       set exception value 1001 to cut-selection.
```

Format 14 - Invoke methods of a Java Class to get System Information

```
      configuration section.
      repository.
          class Sys as "java.lang.System".

      working-storage section.
      77  buffer      pic x(50).

      procedure division.
      main.
          set buffer to Sys:>getProperty("os.name")
          display "Operating System: " buffer.

          set buffer to Sys:>getProperty("os.arch")
          display "OS Architecture: " buffer.

          set buffer to Sys:>getProperty("java.version")
          display "Java Version: " buffer.
```

Format 15 - Get the exact number of items in a nested dynamic capacity table

```
      working-storage section.

      01 group-1.
         03 table-1 occurs dynamic capacity cap-1.
            05 sub-table-1 occurs dynamic capacity cap-1-1.
               07 sub-item-1 pic x(10).

      77 cap pic 9(4).

      PROCEDURE DIVISION.

      MAIN.
          move "something" to sub-item-1(1, 1).
          move "something" to sub-item-1(1, 2).
          move "something" to sub-item-1(2, 1).

          display "highest capacity of sub-table-1 in table-1:" cap-1-1.
     *>   it will display '2'

          set cap to capacity of sub-table-1(1).
          display "capacity of sub-table-1 in table-1(1): " cap.
     *>   it will display '2'

          set cap to capacity of sub-table-1(2).
          display "capacity of sub-table-1 in table-1(2): " cap.
     *>   it will display '1'
```

SORT

Format 1

```
SORT File-Name-1 { { ON {ASCENDING } KEY {Data-Name-1} ... } ... }
                        {DESCENDING}

  { KEY AREA IS Key-Table }

  [ WITH DUPLICATES IN ORDER ]

  [ COLLATING SEQUENCE {IS Alphabet-Name-1 [Alphabet-Name-2] }

  { INPUT PROCEDURE IS Procedure-Name-1 [{THROUGH} Procedure-Name-2] }
                                        {THRU   }
  { USING { File-Name-2 } ...                                       }

  { OUTPUT PROCEDURE IS Procedure-Name-3 [{THROUGH} Procedure-Name-4] }
                                         {THRU   }
  { GIVING { File-Name-3 } ...                                       }
```

Format 2

```
SORT Data-Name-2  [ ON {ASCENDING  }  [KEY Data-Name-1]...]...
                       {DESCENDING }

  [ WITH DUPLICATES IN ORDER ]

  [ COLLATING SEQUENCE {IS Alphabet-Name-1 [Alphabet-Name-2] }
```

Syntax rules

All Formats

1. A SORT statement may appear anywhere in the procedure.
2. Alphabet-Name-1 shall reference an alphabet that defines an alphanumeric collating sequence.
3. Alphabet-Name-2 shall reference an alphabet that defines a national collating sequence.
4. Key-Table must name a data item that is not located in the record for sort-file. Key-table may not be subordinate to an OCCURS clause, nor may it be reference modified.
5. Key-Table must reference a data item whose size is an even multiple of 7. Typically, programs will declare it with a similar format:

```
01  KEY-TABLE.
    03  SORT-KEY OCCURS N TIMES.
        05  KEY-ASCENDING   PIC X  COMP-X.
        05  KEY-TYPE        PIC X  COMP-X.
        05  KEY-OFFSET      PIC XX COMP-X.
        05  KEY-SIZE        PIC XX COMP-X.
        05  KEY-DIGITS      PIC X  COMP-X.
```

Format 1

6. File-name-1 shall be described in a sort-merge file description entry in the data division.
7. 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 describe neither 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.

8. 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. The data items identified by key data-names shall not be variable-length data items.
   D. If file-name-1 has more than one record description, then the data items identified by key data-names need be described in only one of the record descriptions. The same byte 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.
   E. 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.
   F. 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 bytes of the record, where x is the number of bytes of the minimum record size for the file referenced by file-name-1.
9. The words THROUGH and THRU are equivalent.
10. 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.
11. 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 begin at the same byte location within its record and occupy the same number of bytes as the prime record key for that file.
12. 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 describe neither 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-3.
13. If file-name-2 references a relative or an indexed file, its access mode shall be sequential or dynamic.
14. Use the KEY AREA option when you do not know the specifics of the sort key until the program is run. You can use this to allow users to enter sort key specifications. For each key, you must specify the following information:

KEY-ASCENDING	Enter 1 to have an ascending sort sequence, 0 for descending.

KEY-TYPE	Describes the underlying data format. The allowed values are: 0 Numeric edited 1 Unsigned numeric (DISPLAY) 2 Signed numeric (DISPLAY, trailing separate) 3 Signed numeric (DISPLAY, trailing combined) 4 Signed numeric (DISPLAY, leading separate) 5 Signed numeric (DISPLAY, leading combined) 6 Signed COMP-2 7 Unsigned COMP-2 8 Unsigned COMP-3 9 Signed COMP-3 10 COMP-6 11 Signed binary (COMP-1, COMP-4, COMP-X) 12 Unsigned binary (COMP-1, COMP-4, COMP-X) 13 Signed native (COMP-5, COMP-N) 14 Unsigned native (COMP-5, COMP-N) 15 Floating point (FLOAT, DOUBLE) 16 Alphanumeric 17 Alphanumeric (justified) 18 Alphabetic 19 Alphabetic (justified) 20 Alphanumeric edited 21 Not used 22 Group
KEY-OFFSET	Describes the distance (in standard character positions) from the beginning of the sort record to the beginning of the key field. The first field in a sort record is at offset 0.
KEY-SIZE	Describes the size of the key field in standard character positions.
KEY-DIGITS	This is used only for numeric keys. It describes the number of digits contained in the key (counting digits on both sides of the decimal point).

If you provide invalid data in the key-table, results are undefined.

Format 2

15. Data-name-2 may be qualified and shall have an OCCURS clause in its data description entry.
16. Data-name-1 is a key data-name, subject to the following rules:
    A. The data item identified by a key data-name shall be the same as, or subordinate to, the data item referenced by data-name-2.
    B. Key data items may be qualified.
    C. The data items identified by key data-names shall not be variable-length data items.
17. The KEY phrase may be omitted only if the description of the table referenced by data-name-2 contains a KEY phrase.

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 record or table element are equal to the contents of the corresponding key data items associated with one or more other records or table elements, the order of return of these records or the relative order of the contents of these table elements 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 record or table element are equal to the contents of the corresponding key data items associated with one or more other records or table elements, the order of return of these records or the relative order of the contents of these table elements is undefined.

Format 1

5. If the file referenced by file-name-1 contains only fixed-length records, any record in the file referenced by file-name-2 containing fewer character positions than that fixed-length is space filled on the right to that fixed length, 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, as follows:
   A. If there is only one record description entry associated with the file referenced by file-name-2 and that record is described as a national data item or as an elementary data item of usage national and of category numeric, numeric-edited, or boolean, the record is filled with national space characters.
   B. Otherwise, the record is space filled with alphanumeric space characters.
6. 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.
7. 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. If the file referenced by file-name-2 is in an open mode when this phase commences, the results of the execution of the SORT statement are undefined. 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. If the file referenced by file-name-3 is in an open mode when this phase commences, the results of the execution of the SORT statement are undefined. When this phase terminates, the file referenced by file-name-3 is not in the open mode.
8. The input procedure may consist of any procedure needed to create the records that are to be made available

to the sort mechanism by executing RELEASE statements. The range includes all statements that are executed as the result of a transfer of control 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. If the range of the input procedure causes the execution of any MERGE, RETURN, or format 1 SORT statements, the results of the execution of the SORT statement are undefined.

9. If the USING phrase is specified, all the records in the file(s) referenced by file-name-2 are transferred to the file referenced by file-name-1. For each of the files referenced by file-name-2 the execution of the SORT statement causes the following actions to be taken:

   A. The processing of the file is initiated and the initiation is performed as if an OPEN statement with the INPUT phrase is executed.

   B. The logical records are obtained and released to the sort operation. Each record is obtained as if a READ statement with the NEXT phrase, the IGNORING LOCK phrase, and the AT END phrase had been executed. When the at end condition exists for file-name-1, the processing for that file connector is terminated. 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. If the size of the record read from the file referenced by file-name-2 is larger than the largest record allowed in the file description entry for file-name-1, the execution of the SORT statement is terminated. If file-name-1 is specified with variable-length records and the size of the record read from the file referenced by file-name-2 is smaller than the smallest record allowed in the file description entry for file-name-1, the execution of the SORT statement is terminated. If a fatal exception condition exists for file-name-1, the SORT is terminated.

   C. The processing of file-name-1 is terminated. The termination is performed as if a CLOSE statement 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.

      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.

10. The output procedure may consist of any procedure needed to process 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 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. If the range of the output procedure causes the execution of any MERGE, RELEASE, or format 1 SORT statement, the results of the execution of the SORT statement are undefined.

11. If an output procedure is specified, control passes to it after the file referenced by file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism after the last statement in the output procedure. When control passes to that return mechanism, the mechanism provides for the 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.

    NOTE - This return mechanism transfers control from the end of the output procedure and is not associated with the RETURN statement.

12. If the GIVING phrase is specified, all the sorted records are written on the file referenced by file-name-3 as the implied output procedure for the SORT statement. For each of the files referenced by file-name-3, the

execution of the SORT statement causes the following actions to be taken:

A. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase 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. Each record is 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 has 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 had been executed.

   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.

13. If the file referenced by file-name-3 contains only fixed-length records, any record in the file referenced by file-name-1 containing fewer character positions than that fixed-length is space filled on the right to that fixed length, 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, as follows:

    A. If there is only one record description entry associated with the file referenced by file-name-2 and that record is described as a national data item or as an elementary data item of usage national and of category numeric, numeric-edited, or boolean, the record is filled with national space characters.

    B. Otherwise, the record is space filled with alphanumeric space characters.

Format 2

14. The SORT statement sorts the table referenced by data-name-2 and presents the sorted table in data-name-2 either in the order determined by the ASCENDING or DESCENDING phrases, if specified, or in the order determined by the KEY phrase associated with data-name-2.
15. To determine the relative order in which the table elements are stored after sorting, the contents of corresponding key data items are compared according to the rules for comparison of operands in a relation condition, starting with the most significant key data item.

    A. If the contents of the corresponding key data items are not equal and the key is associated with the ASCENDING phrase, the table element containing the key data item with the lower value has the lower occurrence number.

    B. If the contents of the corresponding key data items are not equal and the key is associated with the DESCENDING phrase, the table element containing the key data item with the higher value has the lower occurrence number.

    C. If the contents of the corresponding key data items are equal, the determination is based on the contents of the next most significant key data item.

16. The number of occurrences of table elements referenced by data-name-2 is determined by the rules in the OCCURS clause.
17. If the KEY phrase is not specified, the sequence is determined by the KEY phrase in the data description entry of the table referenced by data-name-2.
18. If the KEY phrase is specified, it overrides any KEY phrase specified in the data description entry of the table referenced by data-name-2.

19. If data-name-1 is omitted, the data item referenced by data-name-2 is the key data item.
20. The sorted table elements of the table referenced by data-name-2 are placed in the table referenced by data-name-2.

Examples

Format 1 - Sort sequential file into another sequential file by alternate key

```
input-output section.
file-control.
  select sort-file
         assign to sort.

  select seq-1-file
         assign to disk "seq1.dat"
         binary sequential
         status is seq-1-status.

  select seq-2-file
         assign to disk "seq2.dat"
         binary sequential.

file section.
fd  seq-1-file.
01  seq-1-record.
    03  seq-1-key                    pic 9(10).
    03  seq-1-alt-key.
        05  seq-1-alt-key-a          pic x(30).
        05  seq-1-alt-key-b          pic 9(10).
    03  seq-1-body                   pic x(50).

fd  seq-2-file.
01  seq-2-record                     pic x(100).

sd  sort-file.
01  sort-record.
    03  sort-key                     pic x(10).
    03  sort-alt-key                 pic x(40).
    03  filler                       pic x(50).

....

procedure division.
main.
  sort sort-file on ascending key sort-alt-key
       using seq-1-file giving seq-2-file.
```

Format 2 - Sort an array of customers

```
working-storage section.
77 i pic 9(3).

01 cust-array occurs 5 times.
   05 cust-code pic x(3).
   05 cust-name pic x(20).
   05 cust-city pic x(20).

procedure division.
main.
  perform fill-array
  perform display-array
  perform sort-array-code
  perform display-array
  perform sort-array-name
  perform display-array
  goback.

fill-array.
  move "444" to cust-code(1)
  move "Adam Smith" to cust-name(1)
  move "New York" to cust-city(1)

  move "222" to cust-code(2)
  move "Eve Lion" to cust-name(2)
  move "Los Angeles" to cust-city(2)

  move "111" to cust-code(3)
  move "Walter Darryn" to cust-name(3)
  move "Chicago" to cust-city(3)

  move "333" to cust-code(4)
  move "Lola Lyn" to cust-name(4)
  move "Washington" to cust-city(4)

  move "555" to cust-code(5)
  move "Will Smith" to cust-name(5)
  move "Riverside" to cust-city(5)
  .

display-array.
  display "------ Customers "
  perform varying i from 1 by 1 until i > 5
    display cust-code(i) " " cust-name(i) " " cust-city(i)
  end-perform
  .
sort-array-code.
  sort cust-array on ascending key cust-code.

sort-array-name.
  sort cust-array on descending key cust-name.
```

START

Format 1

```
START file-name-1 {FIRST}
                  {LAST}
                  { [ KEY {IS relational-operator {data-name-1      } }] }
                                                 {record-key-name-1}
                        {IS FIRST                                    }
                        {IS LAST                                     }
                           [{WITH LENGTH} arithmetic-expression-1] ]
                            {WITH SIZE  }
                  [ WHILE KEY IS relational-operator {data-name-2} ]
                                                     {literal-1}
   [INVALID KEY imperative-statement-1]
     [NOT INVALID KEY inperative-statement-2]
[END-START]
```

Format 2

```
START TRANSACTION
```

Format 3

```
START file-name-1
   KEY IS data-name-3 [ INDEX IS { literal-1   } ]
                                 { data-name-4 }
```

Syntax Rules

1. The access mode of the file referenced by file-name-1 shall be either sequential or dynamic.
2. If the organization of the file referenced by file-name-1 is sequential, either the FIRST or the LAST phrase shall be specified.
3. In the KEY phrase, relational-operator is a relational operator specified in the general-relation format, with the exception of the relational operators 'IS NOT EQUAL TO' or 'IS NOT='.
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 clause in the associated file control entry.
6. For indexed files, data-name-1, if specified, shall reference either:
   A. A data item specified as a prime or alternate record key associated with file-name-1, or
   B. A data item with the following characteristics:
      i. Its leftmost character position within a record of the file corresponds to the leftmost character position of a prime or alternate record key that is associated with file-name-1 and that is defined without the SOURCE phrase in the RECORD KEY clause or ALTERNATE RECORD KEY clause.
      ii. It has the same class, category, and usage as that record key.
      iii. Its length is not greater than the length of that record key.
7. Record-key-name-1 shall be specified with the SOURCE phrase in the RECORD KEY clause or in the ALTERNATE RECORD KEY clause in the file control entry for file-name-1.
8. With the WHILE clause only the LIKE relational operator is admitted. With the KEY clause the following relational operators are admitted: <>, <, <=, >, >=, = (optionally preceded by the NOT word), LESS, NOT LESS,

GREATER, NOT GREATER and EQUAL.

9. If the LENGTH or SIZE phrase is specified, file-name-1 shall reference a file with indexed organization.
10. Data-Name-2 and Literal-1 are regular expressions.
11. START and TRANSACTION are required words.
12. data-name-3 is a data item whose declaration includes an IDENTIFIED BY clause, and is included in the XD record declarations for file-name-1.
13. data-name-4 and literal-1 are numeric values indicating which occurence of the tag name referenced by xml_file_record_node is returned by the next READ NEXT statement.

General Rules

Format 1

1. The open mode of the file connector referenced by file-name-1 shall be input or I-O.
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.
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.
7. Following the unsuccessful execution of a START statement, the file position indicator is set to indicate that no valid record position has been established. For indexed files, the key of reference is undefined.

Relative Files

8. 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.
9. 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. Numeric comparison rules apply.
    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.
10. The comparison uses the data item referenced by the RELATIVE KEY phrase of the ACCESS MODE clause associated with file-name-1.
11. If FIRST is specified, the file position indicator is set to the relative record number of the first existing logical record in the file. If no records exist in the file, the invalid key condition exists and the execution of the START statement is unsuccessful.
12. If LAST is specified, the file position indicator is set to the relative record number of the last existing logical

record in the file. If no records exist in the file, the invalid key condition exists and the execution of the START statement is unsuccessful.

Indexed Files

13. The value of arithmetic-expression-1 reflects the number of characters that will be used as a partial key for positioning to a record in file-name-1. If data-name-1 or record-key-name-1 is of class alphanumeric, arithmetic-expression-1 is the number of alphanumeric character positions; if data-name-1 or record-key-name-1 is of class national, arithmetic-expression-1 is the number of national character positions.
14. If arithmetic-expression-1 does not evaluate to a positive non-zero integer that is less than or equal to the length of the associated key, the I-O status value in the file connector referenced by file-name-1 is set to '23', the invalid key condition exists, and the execution of the START statement is unsuccessful.
15. 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.
16. 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. If the execution of the START statement is successful, this key of reference is used for subsequent sequential READ statements referencing file-name-1.
17. 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 specified in the LENGTH or SIZE clause, if specified, or else 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. Then, either:
        i. 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
        ii. 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.
18. If FIRST is specified, the file position indicator is set to the value of the primary key of the first existing logical record in the physical file and the key of reference is set to the primary key. If no records exist in the file, the I-O status value in the file connector referenced by file-name-1 is set to '23', the invalid key condition exists, and the execution of the START statement is unsuccessful.
19. If LAST is specified, the file position indicator is set to the value of the primary key of the last existing logical record in the physical file and the key of reference is set to the primary key. If no records exist in the file, the I-O status value in the file connector referenced by file-name-1 is set to '23', the invalid key condition exists, and the execution of the START statement is unsuccessful.
20. The implied subject of the WHILE filter in a START statement is the key value of the key of reference in the record that would be accessed by the READ statement. Records with key of reference values that match the specified pattern regular expression are returned during subsequent sequential READ statements. Records with key of reference values that do not match the specified pattern regular expression are skipped during subsequent sequential READ statements. If the word NOT is specified in the WHILE phrase, then filtering is

reversed.

Sequential Files

21. If FIRST is specified, the file position indicator is set to 1 if records exist in the physical file. If no records exist in the file, or the physical file does not support the ability to position at the first record, the I-O status value in the file connector referenced by file-name-1 is set to '23', the invalid key condition exists, and the execution of the START statement is unsuccessful.
22. If LAST is specified, the file position indicator is set to the record number of the last existing logical record in the physical file. If no records exist in the file, or the physical file does not support the ability to position at the last record, the I-O status value in the file connector referenced by file-name-1 is set to '23', the invalid key condition exists, and the execution of the START statement is unsuccessful.

Format2

23. The START TRANSACTION statement identifies the beginning of a transaction. This feature, which invokes the transaction natively implemented by the file system in use, may not work for all file system supported .
24. After the START TRANSACTION, each file update operation is recorded until the next COMMIT or ROLLBACK.
25. If the START TRANSACTION statement fails, the special register TRANSACTION-STATUS will be updated with value "98".

Format 3

26. A Format 3 START statement resets the position of a key in XML syntax. The KEY IS clause resets the current internal counterpart so that a subsequent READ NEXT reads the occurrence of that key specified by data-name-3.
27. If the INDEX clause is omitted, the index is assumed to be 1.
28. Once a START request is performed, the indicated key and all sub-keys are associated with a position, and not with any data in the internal representation.
29. If a START is followed immediately by WRITE KEY, that key is placed immediately before the node that would have been read if a START had been followed with READ NEXT. To add a node after this node, use a READ NEXT statement specifying the same key specified on the START.

Examples

Format 1 - Start on exact key checking for invalid key

```
move 1234 to cust-code
start customers key = cust-code
      invalid key display message "Invalid key!"
      not invalid key display message "Started Ok"
end-start
```

Format 1 - Start on key equal or greater than a value

```
move 956 to cust-code
start customers key not < cust-code
```

Format 1 - Start on first key of an ISAM file

```
start customers key is first
```

Format 1 - Start on last key of an ISAM file

```
start customers key is last
```

Format 1 - Start on first record of a sequential file

```
start customers first
```

Format 1 - Start on last record of a sequential file

```
start customers last
```

Format 1 - Start filtering all records whose primary key value includes "SALT" (case insensitive)

```
start customers-file while key is like "(?i).*salt.*".
```

Format 2 - Start transaction

```
*> This sample will work only with a file system that supports transactions
*> The select should include lock clause with rollback: lock automatic with rollback
...
input-output section.
file-control.
select invoices assign to inv-path
   organization indexed
   access dynamic
   record key cust-code
   lock mode manual with rollback |the rollback clause is required to
   status inv-status.             |activate transaction management
...
procedure division.
...
   start transaction
   display "Customer to apply 10% discount? "
   accept ws-cust-code
   move ws-cust-code to cust-code
   read customers
        invalid key display message "Customer not found"
                    rollback
                    exit paragraph
   end-read
   accept ws-date from date
   move ws-date to cust-last-date-discount
   rewrite cust-rec
   move ws-cust-code to inv-cust-code
   start invoices key = inv-cust-code
         invalid key set end-of-invoices to true
         not invalid key set end-of-invoices to false
   end-start
   perform until end-of-invoices
     read invoices next at end exit perform
          not at end
              if inv-cust-code not = ws-cust-code
                 exit perform
              end-if
     end-read
     if inv-paid-status = "N"
        move 10 to inv-discount
        rewrite invoice-rec
     end-if
   end-perform
   display "Changes applied, confirm Commit: (Y/N)"
   accept apply-commit
   if apply-commit = "Y"
      commit *> All changes are definitely applied
   else
      rollback *> All changes get undone
   end-if.
```

STOP

Format 1

```
STOP RUN [ {RETURNING} {identifier-1} ]
           {GIVING   } {literal-1   }
```

Format 2

```
STOP Literal-1
```

Format 3

```
STOP THREAD [thread-handle]
```

Syntax rules

Formats 1 and 2

1. If a STOP 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.

Format 3

4. thread-handle is the handle of the thread.

General rules

Format 1

1. 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. Any constraints on the value of literal-1 or the contents of the data item referenced by identifier-1 are defined by the implementor.
2. Execution of the run unit terminates and control is transferred to the operating system.

Format 2

3. STOP statement is useful to STOP the execution of the program and pass the control to the isCOBOL Debugger.
   The behavior changes depending on the value of the iscobol.rundebug * configuration property:
   - when the property is set to 0 (default), Literal-1 is displayed on the system output; press Enter to continue program execution,
   - when the property is set to 1, Literal-1 is displayed on the system output, the Debugger is attached but it still doesn't have control; press Enter to give the control to the Debugger
   - when the property is set to 2, the Debugger stops on the STOP statement and has the control.

Format 3

4. STOP THREAD statement is useful to STOP an executed thread.
5. If thread-handle is omitted, the currently executing thread is stopped. If the current thread is the only thread, the STOP THREAD statement behaves like STOP RUN.
6. The STOP THREAD statement needs a YIELD statement to work correctly, else the thread will not be released.

Examples

Format 1 - Exit the program and terminate the run unit

```
stop run
```

Format 2 - Pass the execution to the graphical debugger

```
stop 123
```

Format 3 - Stop a display counter thread

```
working-storage section.
77 thandle  usage handle of thread.
77 counter   pic 9(9).

procedure division.
main.
 perform thread display-counter handle thandle
 perform increase-counter
 stop thread thandle
 display message "done!"
 goback
 .

display-counter.
 perform until 1 = 0
   yield
   display counter
 end-perform.

increase-counter.
 perform 99900000 times
   add 1 to counter
 end-perform.
```

STRING

General Format

```
STRING { { Identifier-1 } ... [ DELIMITED BY { [TRAILING] {Identifier-2} } ] } ...
         { Literal-1    }                    {            {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. All literals shall be described as alphanumeric, boolean, 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.
2. Literal-1 or literal-2 shall not be a figurative constant that begins with the word ALL.
3. Identifier-3 shall not be reference modified.
4. Identifier-3 shall not reference an edited data item and shall not be described with the JUSTIFIED clause. JUSTIFIED destination items are allowed only with the -cm compiler option.
5. Identifier-3 shall not reference a strongly-typed group item.
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' shall 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.
9. If the TRAILING clause is set, the rightmost contiguous character specified by Literal-1 of the source is not transferred to the receiver, Identifier-3.
10. Literal-1 or the data item referenced by identifier-1 is the sending operand. The data item referenced by identifier-3 is the receiving operand.
11. isCOBOL accepts numeric data items that are not USAGE DISPLAY as identifier-1. When identifier-1 is not USAGE DISPLAY the informational error #288 is generated. The memory content of identifier-1 is then used in the STRING statement.

General rules

1. 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.
2. 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.

3. 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.
   B. If the DELIMITED phrase is specified and literal-2 is specified or identifier-2 is specified and is not a zerolength item, 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 and the SIZE phrase is specified or identifier-2 is specified and is a zero-length item, 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.
4. If the POINTER phrase is specified, the data item referenced by identifier-4 shall have a value greater than zero at the start of execution of the STRING statement.
5. 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.
6. 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.
7. 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.
8. 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 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.
   C. If the ON OVERFLOW phrase is not specified, execution continues.
   D. The NOT ON OVERFLOW phrase, if specified, is ignored.
9. If, at the time of execution of a STRING statement with the NOT ON OVERFLOW phrase, the conditions described in general rule 8 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.

10. If identifier-1, or identifier-2, occupies the same storage area as identifier-3, or identifier-4, or if identifier-3 and identifier-4 occupy the same storage area, the result of the execution of this statement is undefined.

Examples

Concatenating 3 variables discarding trailing spaces

```
initialize str-result
string str-1 delimited by trailing spaces
       str-2 delimited by trailing spaces
       str-3
  into str-result
```

Concatenating variables and literals delimited by size and including new lines, validating overflow

```
initialize cust-data
string "Customer Name : " cust-last-name ", " cust-first-name x"0d0a"
       "Customer Age  : " cust-age x"0d0a"
       "Customer Address : " cust-address
  into cust-data
    on overflow display message "Customer data is too large!"
end-string
```

Using pointer to indicate where to start concatenating

```
*> my-str is defined as pic x(50)
initialize my-str
move 1 to idx
string "hello " "world!" into my-str pointer idx
move 25 to idx
string "good bye!" into my-str pointer idx
move 35 to idx
string "hi again!" into my-str pointer idx
*> Result in my-str: hello world!            good bye! hi again!
```

SUBTRACT

Format 1

```
SUBTRACT { Indentifier-1 } ... FROM { Identifier-2 [ROUNDED] } ...
         { Literal-1     }
         { Class-1       }

  [ ON SIZE ERROR Imperative-Statement-1 ]

  [ NOT ON SIZE ERROR Imperative-Statement-2 ]

  [END-SUBTRACT]
```

Format 2

```
SUBTRACT { Indentifier-1 } ... FROM { Identifier-2 } ...
         { Literal-1     }          { Literal-2    }
         { Class-1       }          { Class-2      }

  GIVING { Identifier-3 [ROUNDED] } ...

  [ ON SIZE ERROR Imperative-Statement-1 ]

  [ NOT ON SIZE ERROR Imperative-Statement-2 ]

  [END-SUBTRACT]
```

Format 3

```
SUBTRACT { CORRESPONDING }{ {Identifier-4} } ... FROM { {Identifier-5} } ...
         { CORR          }

  [ ON SIZE ERROR Imperative-Statement-1 ]

  [ NOT ON SIZE ERROR Imperative-Statement-2 ]

  [END-SUBTRACT]
```

Syntax rules

Formats 1 and 2

1. Identifier-1 and identifier-2 shall reference numeric data items.
2. Literal-1 and literal-2 shall be numeric literals.
3. Class-1 and Class-2 shall be OBJECT-REFERENCE of Java numeric primitive types (i.e. "int") or objects (i.e. "java.lang.Integer")

Format 2

4. Identifier-3 shall reference a numeric data item or a numeric-edited data item.

Format 3

5. CORRESPONDING and CORR are synonymous.
6. Identifier-4 and identifier-5 shall reference a group item.

General rules

1. When format 1 is used, the initial evaluation consists 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 is 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 is 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 SUBTRACT statement. The result of the subtraction from the value of each data item referenced by identifier-2 is equivalent to the result of the arithmetic expression

   (identifier-2 – initial-evaluation)

   where initial-evaluation represents the result of the initial evaluation.
2. When format 2 is used, the initial evaluation consists 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. The result is stored as the new value of the data item referenced by identifier-3.

   When standard arithmetic is in effect, the result of the initial evaluation is equivalent to the result of the arithmetic expression

   (operand-2 – (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 SUBTRACT statement and the value of operand-2 is the value of either literal-2 or the data item referenced by identifier-2 in the SUBTRACT statement.
3. In Format 3, elementary numeric items in Identifier-4 are subtracted from the corresponding items in Identifier-5. The values are then stored in Identifier-5.

Examples

Format 1 - Subtract several variables and literals from variable

```
move 100 to num-result
move 10  to num-1
move 5   to num-2
subtract num-1 num-2 10 from num-result
*> new value for num-result : 75
```

Format 2 - Subtract several numbers from number and leave result on another variable validating size error

```
subtract 10 20 30 from 100
  giving num-result
  on size error display message "Size error on num-result!"
end-subtract
*> new value for num-result : 40
```

Format 3 - Subtract values of one group data item from another group with corresponding sub-items

```
working-storage section.
01 work-hours.
   05 test-hours    pic 9(3) value 10.
   05 doc-hours     pic 9(3) value 11.
   05 support-hours pic 9(3) value 12.
01 notconfirmed-work-hours.
   05 test-hours    pic 9(3) value 2.
   05 doc-hours     pic 9(3) value 4.
   05 support-hours pic 9(3) value 2.

procedure division.
add-hours.
   subtract corresponding notconfirmed-work-hours from work-hours
       on size error display message "Error subtracting from work-hours"
       not on size error
           display message
                   test-hours    of work-hours " , "
                   doc-hours     of work-hours " , "
                   support-hours of work-hours
   end-subtract
```

SYNCHRONIZED

General Format

```
SYNCHRONIZED

  [ ON Sync-Var ]

    [Statement-1]
    [Statement-2]
    [Statement-n]

  [END-SYNCHRONIZED]
```

Syntax Rules

1. Statement-1 through statement-n represent either one or more imperative statements or a conditional statement optionally preceded by one or more imperative statements.
2. Sync-Var is any item defined in the Data Division.

General rules

1. SYNCHRONIZED statement allows you to synchronize a block of statements when running in mutithread environment. When a thread is executing the statements of a synchronized block, other threads instructed to execute the same block of statements block will wait for the first thread to finish.
2. When the ON clause is specified, Sync-Var becomes the lock, otherwise the block itself is the lock. Two blocks that are synchronized on the same Sync-Var can't be executed at the same time.

Examples

Scenario 1 - Multiple threads in the same program.

The following code may set the item *varx* to a unexpected value because it's not safe to access the same data items from different threads:

```
working-storage section.
77 t1     handle of thread.
77 t2     handle of thread.
77 t3     handle of thread.
77 varx   pic XX.
procedure division.
main.
   move "AA" to varx
   perform thread p1 handle t1.
   perform thread p2 handle t2.
   perform thread p3 handle t3.
   ACCEPT OMITTED
   STOP RUN.
p1.
    perform UNTIL 1 = 2
      MOVE "AA" TO varx
    end-perform.
p2.
    perform UNTIL 1 = 2
      MOVE "BB" TO varx
    end-perform.
p3.
    perform UNTIL 1 = 2
      IF varx = "AA" or "BB"
         continue
      ELSE
         display "Unexpected condition! varx=" varx
         STOP RUN
      END-IF
    end-perform.
```

Synchronizing the access to *varx* as follows, resolves the issue:

```
working-storage section.
77 t1     handle of thread.
77 t2     handle of thread.
77 t3     handle of thread.
77 varx   pic XX.
procedure division.
main.
   move "AA" to varx
   perform thread p1 handle t1.
   perform thread p2 handle t2.
   perform thread p3 handle t3.
   ACCEPT OMITTED
   STOP RUN.
p1.
    perform UNTIL 1 = 2
      SYNCHRONIZED
      MOVE "AA" TO varx
      END-SYNCHRONIZED
    end-perform.
p2.
    perform UNTIL 1 = 2
      SYNCHRONIZED
      MOVE "BB" TO varx
      END-SYNCHRONIZED
    end-perform.
p3.
    perform UNTIL 1 = 2
      SYNCHRONIZED
      IF varx = "AA" or "BB"
         continue
      ELSE
         display "Unexpected condition! varx=" varx
         STOP RUN
      END-IF
      END-SYNCHRONIZED
    end-perform.
```

The following is a similar case that reproduces a different problematic condition: instead of setting items to an undefined value, it generates an internal error for concurrent access to the same group data item:

```
working-storage section.
77 t1     handle of thread.
77 t2     handle of thread.
77 t3     handle of thread.
01 vars.
   03 var1   pic 9(3).
   03 var2   pic 9(5).
procedure division.
main.
   move 1 to var1
   move 2 to var2
   perform thread P1 handle t1.
   perform thread P2 handle t2.
   perform thread P3 handle t3.
   ACCEPT OMITTED
   STOP RUN.
p1.
    perform UNTIL 1 = 2
      MOVE 1 TO var1
    end-perform.
p2.
    perform UNTIL 1 = 2
      MOVE 2 TO var2
    end-perform.
p3.
    perform UNTIL 1 = 2
      add 1 to var1
      add 1 to var2
    end-perform.
```

Also in this case, using the SYNCHRONIZED statement will fix the issue:

```
working-storage section.
77 t1     handle of thread.
77 t2     handle of thread.
77 t3     handle of thread.
01 vars.
   03 var1   pic 9(3).
   03 var2   pic 9(5).
procedure division.
main.
   move 1 to var1
   move 2 to var2
   perform thread P1 handle t1.
   perform thread P2 handle t2.
   perform thread P3 handle t3.
   ACCEPT OMITTED
   STOP RUN.
p1.
    perform UNTIL 1 = 2
      SYNCHRONIZED
      MOVE 1 TO var1
      END-SYNCHRONIZED
    end-perform.
p2.
    perform UNTIL 1 = 2
      SYNCHRONIZED
      MOVE 2 TO var2
      END-SYNCHRONIZED
    end-perform.
p3.
    perform UNTIL 1 = 2
      SYNCHRONIZED
      add 1 to var1
      add 1 to var2
      END-SYNCHRONIZED
    end-perform.
```

Scenario 2 - Multiple threads identified by different programs.

Running PROGA that calls in thread PROGB, you will obtain an internal error because the two programs are accessing the same group data item at the same time:

```
program-id. proga.
working-storage section.
77 t1     handle of thread.
01 vars.
   03 var1   pic 9(3).
   03 var2   pic 9(5).
procedure division.
main.
   move 1 to var1
   move 2 to var2
   call thread "progb" handle in t1 using vars.
   call "c$sleep" using 1
   perform para2.
   ACCEPT OMITTED
   STOP RUN.
para2.
    perform UNTIL 1 = 2
      MOVE 2 TO var1
      MOVE 2 TO var2
    end-perform.
```

```
program-id. progb.
working-storage section.
linkage section.
01 vars.
   03 var1   pic 9(3).
   03 var2   pic 9(5).
procedure division using vars.
main.
    perform UNTIL 1 = 2
      MOVE 1 TO var1
      MOVE 1 TO var2
    end-perform.
    goback.
```

In this case it's not enough to synchronize the problematic code in the two programs. Since the two threads are in separate programs, the program itself can't be the synchronizing object. You need a third item, a class that shares a lock for both programs, e.g.

```
identification division.
class-id. myclass as "myclass".

identification division.
factory.
working-storage section.
public.
77 MYLOCK  pic x.

procedure division.

end factory.
```

```
program-id. proga.
configuration section.
repository.
    class myclass as "myclass".
working-storage section.
77 t1     handle of thread.
01 vars.
   03 var1   pic 9(3).
   03 var2   pic 9(5).
procedure division.
main.
   move 1 to var1
   move 2 to var2
   call thread "progb" handle in t1 using vars.
   call "c$sleep" using 1
   perform para2.
   ACCEPT OMITTED
   STOP RUN.
para2.
    perform UNTIL 1 = 2
     SYNCHRONIZED on myclass:>MYLOCK
      MOVE 2 TO var1
      MOVE 2 TO var2
     END-SYNCHRONIZED
    end-perform.
```

```
program-id. progb.
configuration section.
repository.
    class myclass as "myclass".
working-storage section.
linkage section.
01 vars.
   03 var1   pic 9(3).
   03 var2   pic 9(5).
procedure division using vars.
main.
    perform UNTIL 1 = 2
     SYNCHRONIZED on myclass:>MYLOCK
       MOVE 1 TO var1
       MOVE 1 TO var2
     END-SYNCHRONIZED
    end-perform.
    goback.
```

TERMINATE

General Format

```
TERMINATE report-name-1
```

Syntax Rules

1. Report-name-1 shall be defined by a report description entry in the report section.
2. If report-name-1 is defined in a containing program, the file description entry associated with report-name-1 shall contain a GLOBAL clause.

General Rules

1. The TERMINATE statement may be executed only for a report that is in the active state. If the report is not in the active state, the EC-REPORT-INACTIVE exception condition is set to exist and the execution of the statement has no other effect.
2. If no GENERATE statement has been executed for a report during the interval between the execution of an INITIATE statement and a TERMINATE statement for that report, the TERMINATE statement causes no processing of any kind to take place for any report groups and has the sole effect of changing the state of the report to inactive.
3. If at least one GENERATE statement has been executed for a report during the interval between the execution of an INITIATE statement and a TERMINATE statement for that report, the TERMINATE statement causes the following actions to take place:
   A. The contents of any control data items are changed to their prior values.
   B. Each control footing is printed, if defined, beginning with the minor control footing, as defined for the GENERATE statement, as though a control break has been sensed in the most major control data item.
   C. The report footing is printed, if defined.
   D. The contents of any control data items are restored to the values they had at the start of execution of the TERMINATE statement.
4. The result of executing a TERMINATE statement in which more than one report-name-1 is specified is as though a separate TERMINATE statement had been executed for each report-name-1 in the same order as specified in the statement. If an implicit TERMINATE statement results in the execution of a declarative procedure that executes a RESUME statement with the NEXT STATEMENT phrase, processing resumes at the next implicit TERMINATE statement, if any.
5. If a non-fatal exception condition is raised during the execution of a TERMINATE statement, execution resumes at the next report item, line, or report group, whichever follows in logical order.
6. The TERMINATE statement does not close the file associated with report-name-1.

Examples

A code snippet from the REPORT-SECTION.cbl sample program installed with isCOBOL under sample/data-access/files

```
...
       report section.
       rd  my-report
           controls are group-name
           page limit is 22
           heading 1
           first detail 4
           last detail 18
           footing 20.
...
       PROCEDURE DIVISION.
...
           terminate my-report.
...
```

TRANSFORM

General Format

```
TRANSFORM identifier-1 CHARACTERS FROM pattern-1 TO pattern-2
```

General rules

OS/VS COBOL supported the TRANSFORM statement and has been replaced by INSPECT CONVERTING statements. isCOBOL supports either statement.

The following OS/VS COBOL TRANSFORM statement:

```
77  DATA-T     PICTURE X(9) VALUE "ABCXYZCCC"
  . . .
    TRANSFORM identifier-1 DATA-T FROM "ABC" TO "CAT"
```

TRANSFORM evaluates each character, changing each A to C, each B to A, and each C to T.

After the TRANSFORM statement is executed. identifier-1 DATA-T contains "CATXYZTTT".

The -cv compiler option is required to compile this statement.

TRY

The TRY statement is used as an exception handler.

General Format

```
TRY Imperative-Statement-1

  [ CATCH {Exception-Class} Imperative-Statement-2 ] ...

  [ CATCH EXCEPTION Imperative-Statement-3 ]

  [ FINALLY Imperative-Statement-4 ]

  END-TRY
```

Syntax rules

1. Imperative-Statement-1, Imperative-Statement-2, Imperative-Statement-3, and Imperative-Statement-4 represent either one or more imperative statements or a conditional statement optionally preceded by one or more imperative statements.
2. Exception-Class is a User-defined word, as defined in the Definitions section in the Preface of this document.
3. Imperative-Statement-1, Imperative-Statement-2, Imperative-Statement-3, and Imperative-Statement-4 may each contain a TRY statement. In this case, the TRY statement is said to be nested.
4. At least a CATCH clause must be specified.
5. The FINALLY clause can be omitted.

General rules

1. Exception-Class must appear in the REPOSITORY Paragraph and reference a class of type java.lang.Trowable.
2. The class referenced by Exception-Class must be trown by Imperative-Statement-1. Consult the javadoc of the methods used by Imperative-Statement-1 for a list of thrown exceptions.
3. A try statement executes one or more imperative statements. If an exception is thrown and the try statement has one or more CATCH clauses that can catch it, then control will be transferred to the first such catch clause.
4. If the FINALLY clause is specified, then Imperative-Statement-4 is executed, no matter whether the try block completes normally or abruptly, and no matter whether a catch clause is first given control.
5. Separate lines of code that might throw an exception can be put into their own TRY blocks, or all the potential problem code can be put into the same TRY block and managed by separate handlers.
6. To associate an exception handler with a TRY block, you must use a CATCH clause.
7. The CATCH EXCEPTION clause catches the generic java.lang.Exception.
8. Imperative-Statement-2 and Imperative-Statement-3 can take advantage of the EXCEPTION-OBJECT internal object, that is automatically set by the runtime, in order to retrieve deatails such as the error message or the exception stack.

Examples

The following code snippets are utilities that checks if a given file exists.

The first snippet catches the generic exception and uses EXCEPTION-OBJECT to advise the user of the error encountered:

```
       CONFIGURATION SECTION.
       REPOSITORY.
           CLASS JFILE AS "java.io.File"
           .

       WORKING-STORAGE SECTION.
       77  W-STATUS PIC 9(9).
       88  FILE-NOTFOUND   VALUE 0.
       88  FILE-EXISTS     VALUE 1.
       88  ERROR-CONDITION VALUE 2.

       LINKAGE SECTION.
       77  FILENAME PIC X(256).

       PROCEDURE DIVISION USING FILENAME.
       MAIN.
           TRY
             IF JFILE:>new(FILENAME):>exists()
                SET FILE-EXISTS TO TRUE
             ELSE
                SET FILE-NOTFOUND TO TRUE
             END-IF
           CATCH EXCEPTION
             SET ERROR-CONDITION TO TRUE
             DISPLAY MESSAGE EXCEPTION-OBJECT:>getMessage()
           FINALLY
             GOBACK W-STATUS
            END-TRY.
```

The second snippet catches the SecurityException thrown by the exists() method of java.io.File, as described in the method javadoc: https://docs.oracle.com/javase/8/docs/api/java/io/File.html#exists--:

```
       CONFIGURATION SECTION.
       REPOSITORY.
           CLASS JFILE  AS "java.io.File"
           CLASS SECEXC AS "java.lang.SecurityException"
           .

       WORKING-STORAGE SECTION.
       77  W-STATUS PIC 9(9).
       88  FILE-NOTFOUND   VALUE 0.
       88  FILE-EXISTS     VALUE 1.
       88  SECURITY-ERROR  VALUE 2.

       LINKAGE SECTION.
       77  FILENAME PIC X(256).

       PROCEDURE DIVISION USING FILENAME.
       MAIN.
           TRY
             IF JFILE:>new(FILENAME):>exists()
                SET FILE-EXISTS TO TRUE
             ELSE
                SET FILE-NOTFOUND TO TRUE
             END-IF
           CATCH SECEXC
             SET SECURITY-ERROR TO TRUE
             DISPLAY MESSAGE EXCEPTION-OBJECT:>getMessage()
           FINALLY
             GOBACK W-STATUS
           END-TRY.
```

UNLOCK

Format 1

```
UNLOCK {File-Name-1} ALL {RECORD }
                         {RECORDS}
```

Format 2

```
UNLOCK ALL {RECORD }
           {RECORDS}
```

Syntax rules

1. File-name-1 shall not refer to a sort file or a merge file.
2. `RECORD` and `RECORDS` are synonymous.

General rules

Format 1

1. Any record locks associated with the file connector referenced by file-name-1 are released by the 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.
2. File-name-1 shall reference a file connector in the open mode.
3. 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

4. This format releases all records locked by all the open files in that program.
5. The execution of the UNLOCK ALL is always successfull, the I-O status is not updated, and no declaratives are executed.

Examples

Format 1 - Unlock all records of one file

```
unlock customers
```

Format 2 - Unlock all records of all files

```
unlock all
```

UNSTRING

General Format

```
UNSTRING Identifier-1

  [ DELIMITED BY [ALL] {Identifier-2} { [ OR [ALL] {Identifier-3} ] } ... ]
                       {Literal-1   }              {Literal-2   }

  INTO { Identifier-4 [ DELIMITER IN Identifier-5 ] [ COUNT IN Identifier-6 ] } ...

  [ WITH POINTER Pointer-Var-1 ]

  [ TALLYING IN Identifier-7 ]

  [ 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 of category alphanumeric or 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. The -cm option removes this restriction. Numeric items shall not be specified with the symbol 'P' in their picture character-string.
5. Identifier-6 and Identifier-7 shall reference integer data items. The symbol 'P' shall not be used in the picture character-string.
6. Pointer-Var-1 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' shall not be used in the picture character-string of Pointer-Var-1.
7. The DELIMITER IN phrase and the COUNT IN phrase may be specified only if the DELIMITED BY phrase is specified.
8. The data item referenced by identifier-1 is the sending operand.
9. The data item referenced by identifier-4 is the receiving operand for data. The data item referenced by identifier-5 is the receiving operand for delimiters.

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. If the data item referenced by identifier-1 is a zero-length item, execution of the UNSTRING statement terminates immediately.
3. Literal-1 or the data item referenced by identifier-2 specifies a delimiter.
4. 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).
5. The data item referenced by Pointer-Var-1 contains a value that indicates a relative character position within the area referenced by identifier-1.
6. The data item referenced by Identifier-7 is a counter that is incremented by 1 for each occurrence of the data item referenced by identifier-4 accessed during the UNSTRING operation.
7. 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 11d.
8. 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.
9. 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. If the data item referenced by identifier-2 or identifier-3 is a zero-length item, that delimiter is ignored. When neither literal-1 nor literal-2 is specified and all data items referenced by identifier-2 and identifier-3 are zero-length items, it is as if the DELIMITED phrase were not specified.

10. 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.

11. 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 Pointer-Var-1. 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 9.) 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 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 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. lf 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 11b through 11f 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.

12. The initialization of the contents of the data items associated with the POINTER phrase or the TALLYING phrase is the responsibility of the user.

13. The content of the data item referenced by Pointer-Var-1 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 Pointer-Var-1 will contain a value equal to the initial value plus the number of characters examined in the data item referenced by identifier-1.

14. When the execution of an UNSTRING statement with a TALLYING phrase is completed, the content of the data item referenced by Identifier-7 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.

15. Either of the following situations causes an overflow condition:
    A. An UNSTRING is initiated, and the value in the data item referenced by Pointer-Var-1 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.
16. When an overflow condition exists, the following occurs:
    A. The UNSTRING operation is terminated.
    B. If the ON OVERFLOW phrase is specified, control is transferred to imperative-statement-1 and execution continues according to the rules for each statement 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.
    C. If the ON OVERFLOW phrase is not specified, execution continues.
    D. The NOT ON OVERFLOW phrase, if specified, is ignored.
17. If, at the time of execution of an UNSTRING statement, the conditions described in general rule 15 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.
18. If identifier-1, identifier-2, or identifier-3, occupies the same storage area as identifier-4, identifier-5, identifier-6, Pointer-Var-1, or Identifier-7, or if identifier-4, identifier-5, or identifier-6, occupies the same storage area as Pointer-Var-1 or Identifier-7, or if Pointer-Var-1 and Identifier-7 occupy the same storage area, the result of the execution of this statement is undefined.

Examples

Get words from a full string and put them into an array, words are separated by "," or "|"

```
working-storage section.
01 ws-str     pic x(50).
01 words      pic x(10) occurs 10 times.
01 str-idx    pic 9(3).
01 word-idx   pic 9(3).
01 word-count pic 9(3).

procedure division.
main.
   move "one|two,apple,pear|peach|last" to ws-str
   move 0 to word-count
   move 1 to str-idx word-idx
   perform until str-idx > 50
     unstring ws-str delimited by "|" or ","
         into words(word-idx)
      pointer str-idx
      tallying word-count
     end-unstring
     add 1 to word-idx
   end-perform.
   perform varying word-idx from 1 by 1 until word-idx > word-count
     display words(word-idx)
   end-perform
   display word-count.
*> word-count will be 6
*> every words(i) will contain one word only
```

WAIT

General Format

```
WAIT FOR { THREAD Thread-Id }
         { LAST THREAD      }

  [Remaining-Phrase]

  [ ON EXCEPTION Imperative-Statement-1 ]

  [ NOT ON EXCEPTION Imperative-Statement-2 ]

  [END-WAIT]
```

Remaining-Phrases are optional and can appear in any order.

```
  { BEFORE TIME Timeout }
  { TEST ONLY           }

  THREAD IN Thread-2

  SIZE IN Size-Item

  STATUS IN status-item
```

Syntax rules

1. Thread-Id and Thread-2 are usage HANDLE or HANDLE OF THREAD data items.
2. Size-Item and, Status-Item are numeric data items.
3. Size-Item, Status-Item and Thread-2 cannot be indexed or reference modified.
4. Status-Item is a two-character group item defined as, PIC X(02), or PIC 9(02).

General rules

1. The WAIT statement waits for a thread to terminate or send a message. The thread used to wait is one of the following:
   A. FOR THREAD thread-ID specifies the thread identified by thread-ID.
   B. FOR LAST THREAD specifies the last thread.
2. If a message is available when the WAIT statement executes, then WAIT statement finishes immediately.
3. When BEFORE TIME is specified, the WAIT statement will time-out after the specified hundredths of seconds. If this happens, the destination item (dest-item) is not updated. If timeout is zero, then the WAIT statement times out immediately if a message is not available. Specifying TEST ONLY is equivalent to specifying a timeout value of zero.
4. The thread ID of the sending or terminating thread is put into thread-2 by WAIT.
5. The size of the message sent is placed in size-item.
6. The status of the WAIT statement is placed in status-item as defined below:

"00"	Success - message received

"10"	Exception - thread does not exist or terminated
"99"	Exception - timed-out

7. There is no control in Size Item and Status Item, the result is setted in according with MOVE statment.
8. If the WAIT statement is successful, statement-2 executes otherwise, statement-1 executes.

Examples

Wait for a thread to finish, in this sample the other thread has its handle in loop-thread-handle

```
wait for thread loop-thread-handle
     on exception
       display message " Loop thread finished, finishing this thread now"
       goback
end-wait
```

WRITE

Format 1

```
WRITE { Record-Name-1    } [ FROM {Identifier-1}  ]
      { FILE File-Name-1 }        {Literal-1   }

  [ {BEFORE} ADVANCING { Identifier-2    [LINE ] } ]
    {AFTER }           { Integer-1       [LINES] }
                       { PAGE                    }

  [ AT {END-OF-PAGE} Imperative-Statement-1 ]
       {EOP        }

  [ NOT AT {END-OF-PAGE} Imperative-Statement-2 ]
           {EOP        }

  [END-WRITE]
```

Format 2

```
WRITE { Record-Name-1    } [ FROM {Identifier-1}  ]
      { FILE File-Name-1 }        {Literal-1   }

  [ WITH LOCK ]

  [ INVALID KEY Imperative-Statement-1 ]

  [ NOT INVALID KEY Imperative-Statement-2 ]

  [END-WRITE]
```

Format 3

```
WRITE { Record-Name-1    } [ FROM {Identifier-1} ] [ WITH NO {CONTROL   } ] [
SIZE Identifier-3 ]
      { FILE File-Name-1 }        {Literal-1   }            {CONVERSION}
```

Format 4

```
WRITE Record-Name-1
  [ KEY IS { ALL data-name-1                          }
           { {PROCESSING-INSTRUCTION } { data-name-2 }}
           { { PLAIN-TEXT            } { literal-1   }}
```

Syntax rules

1. The write file is the file referenced by file-name-1 or by the file-name associated with record-name-1.
2. If the organization of the write file is sequential, format 1 shall be specified.
3. If the organization of the write file is indexed or relative, format 2 shall be specified.
4. 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.
5. Record-name-1 is the name of a logical record in the file section of the data division and may be qualified.
6. 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.
7. If the FILE phrase is specified, the FROM phrase shall also be specified and identifier-1 shall be valid as a sending operand in a MOVE statement;
8. If the FILE phrase is specified, the description of identifier-1, including its subordinate data items, shall not contain a data item described with a USAGE OBJECT REFERENCE clause.
9. Identifier-2 shall reference an integer data item.
10. Integer-1 shall be positive or zero.
11. The phrases ADVANCING PAGE and END-OF-PAGE shall not both be specified in a single WRITE statement.
12. 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 associated with the write file.
13. If record-name-1 is defined in a containing program and is referenced in a contained program, the file description entry for the file-name associated with record-name-1 shall contain a GLOBAL clause.
14. If automatic locking has been specified for the write file, neither the WITH LOCK phrase nor the WITH NO LOCK phrase shall be specified.
15. END-OF-PAGE and EOP are synonymous.
16. Identifier-3 is a numeric data item or literal.
17. Record-name-1 is the name of an 01 level group item defined in an XD record declaration within the Data Division .
18. Data-name-1 can be qualified, and references a data item whose declaration includes an IDENTIFIED BY clause and is included in the XD record declarations for file-name-1.

General rules

All Files

1. The write file connector is the file connector associated with the write file. If the access mode of the write file is sequential, the open mode of the write file connector shall be extend or output. Otherwise, the open mode

of the write file connector shall be I-O or output. If the open mode is not as described, the execution of the WRITE statement is unsuccessful and the I-O status associated with file-name-1 is set based on your iscobol.file.status * setting.

2. 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.

3. The figurative constant SPACE when specified in the WRITE statement references one alphanumeric space character.

4. The result of execution of a WRITE statement with the FILE phrase is equivalent to the execution of the following in the order specified:

   — The statement:

      MOVE identifier-1 TO implicit-record-1

      or

      MOVE literal-1 TO implicit-record-1

   — 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 references an intrinsic function, as though implicit-record-1 were a record description entry subordinate to the file description entry having the same class, category, usage, and length as the returned value of the intrinsic function, or

   B. when identifier-1 does not reference an intrinsic function, 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 class, category, usage, and length as literal-1.

5. If the locking mode of the write file connector is single record locking, any record lock associated with that file connector is released by the execution of the WRITE statement.

6. If record locks have an effect for the write file connector and the WITH LOCK phrase is specified, the record lock associated with the record written is set when the execution of the WRITE statement is successful.

7. The file position indicator is not affected by the execution of a WRITE statement.

8. The execution of a WRITE statement causes the value of the I-O status of the write file connector to be updated.

9. The successful execution of a WRITE statement releases a logical record to the operating environment.

NOTE - Logical records in relative and sequential files may have a length of zero. Logical records in an indexed file shall always be long enough to contain the record keys.

10. When record-name-1 is specified, if the number of bytes to be written to the file is greater than the number of bytes in record-name-1, the content of the bytes that extend beyond the end of record-name-1 are undefined.
11. If the execution of a WRITE statement is unsuccessful, the write operation does not take place, the content of the record area is unaffected, and the I-O status of the write file connector is set to a value indicating the cause of the condition as specified in the following general rules.

Sequential and Print Files

12. The successor relationship of a sequential file is established by the order of execution of WRITE statements when the physical file is created. The relationship does not change except when records are added to the end of a physical file.
13. When the organization of the write file connector is sequential and the open mode is extend, the execution of the WRITE statement will add records to the end of the physical file as though the open mode of the file connector were output. If there are records in the physical 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 physical file.
14. If two or more file connectors for a sequential file add records by sharing the physical file after opening it in extend mode, the added records follow the records present in the physical file when it was opened, but are otherwise in an undefined order.
15. When an attempt is made to write beyond the externally-defined boundaries of the physical file, the execution of the WRITE statement is unsuccessful and the I-O status value associated with file-name-1 is set based on your iscobol.file.status * setting.
16. Both the ADVANCING phrase and the END-OF-PAGE phrase allow control of the vertical positioning of each line on a representation of a printed page. If the ADVANCING phrase is not used, automatic advancing shall be provided by the implementor to act as if the user has specified AFTER ADVANCING 1 LINE. If the physical file does not support vertical positioning, the ADVANCING and END-OF-PAGE phrases are ignored. If the physical file does support vertical positioning and the ADVANCING phrase is specified, 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 page advancement specified occurs after Record-Name-1 is added to the file.
    E. If the AFTER phrase is used, the page advancement specified occurs before Record-Name-1 is added to the file.
    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. If physical page has no meaning in conjunction with a specific device, advancing will be provided as if the user had specified BEFORE or AFTER (depending on the phrase used) ADVANCING 1 LINE.
17. Both the ADVANCING phrase and the END-OF-PAGE phrase causes the file to be treated as a print file.
18. If the LINAGE clause is specified in the file description entry of the associated file, an end-of-page condition

occurs when the lines written by a WRITE statement do not fit within the current page body. This occurs when:

A. The logical end of the representation of the printed page is reached. This occurs when the associated LINAGE-COUNTER is equal to or exceeds the page size. If the AFTER phrase is specified or implied, the device is repositioned to the first line that may be written on the next logical page and the logical record is presented on that line. If the BEFORE phrase is specified, the logical record is presented and the device is repositioned to the first line that may be written on the next logical page.

B. The FOOTING phrase is specified in the LINAGE clause and the execution of the WRITE statement causes printing or spacing within the footing area of a page body. This occurs when the associated LINAGE-COUNTER is equal to or exceeds the current value of the footing start and is less than the page size.

19. When an end-of-page condition occurs, the following actions take place:

    A. If the END-OF-PAGE 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 WRITE statement. The NOT END-OF-PAGE phrase, if specified, is ignored.

20. 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 and 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 execution of imperative-statement-1, control is transferred to the end of the WRITE statement. The END-OF-PAGE phrase, if specified, is ignored.

21. When the SIZE clause is specified, if writing on a disk file, the file must be open in EXTEND mode. Identifier-3 controls how many bytes are flushed to the file.

22. In print files trailing spaces are removed from the record.

23. When the NO CONTROL clause is specified, line feed characters are removed.

24. When the NO CONVERSION clause is specified, line feed characters are removed and trailing spaces are kept.

Relative Files

25. The WRITE statement proceeds as follows:

    A. If the access mode of the write file connector is sequential, the successful execution of the WRITE statement causes a record to be released to the operating environment. If the open mode of the write file connector is output, the first record released after the OPEN is 1. If the open mode is extend, the first record released after the OPEN is assigned a record number that is one greater than the highest relative record number existing in the physical file. Subsequent records released have relative record numbers that are ascending ordinal numbers. If the physical file is shared and the open mode is extend, the record numbers are not necessarily consecutive. Otherwise, they are consecutive. 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 operating environment during execution of the WRITE statement according to the rules for the MOVE statement. If the maximum numeric value allowed for the relative key data item is exceeded by the relative record number that would be generated by a successful WRITE operation, the WRITE statement is unsuccessful, and the I-O status associated with file-name-1 is set based on your iscobol.file.status * setting.

    B. If the access mode of the write file connector 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 runtime element with the relative record number to be associated with the record that is to be written. If there is no record in the physical file with a relative record number that matches the relative key value, the

record is released to the operating environment. If a record in the file matches, the execution of the WRITE statement is unsuccessful, the invalid key condition exists, and the I-O status associated with file-name-1 is set based on your iscobol.file.status * setting.

26. 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 operating environment has a relative record number one greater than the highest relative record number existing in the file. Subsequent records released to the operating environment 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 operating environment during execution of the WRITE statement according to the rules for the MOVE statement.
27. 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.
28. 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 runtime element with the relative record number to be associated with the record that is to be written. That record is then released to the operating environment by the execution of the WRITE statement.
29. The invalid key condition exists under the following circumstances, regardless of any locks that are associated with the record being accessed:
    A. When the value of the relative key data item specifies a record that already exists in the file, the I-O status associated with file-name-1 is set based on your iscobol.file.status * setting.
    B. When an attempt is made to write beyond the externally defined boundaries of the file, the I-O status associated with file-name-1 is set based on your iscobol.file.status * setting.

Indexed Files

30. Successful execution of a WRITE statement causes the content of the record area to be released. The operating environment 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.
31. The comparison used for ensuring uniqueness or for ordering records in the physical file is based on the collating sequence for the file according to the rules for a relation condition.
32. The value of the prime record key shall not be equal to the value of the prime record key of any record existing in the file.
33. The data item specified as the prime record key shall be set by the runtime element to the desired value prior to the execution of the WRITE statement.
34. If the access mode of the write file connector is sequential, records shall be released to the operating environment through that file connector in ascending order of prime record key values according to the collating sequence of the file. When the open mode is extend, the first record released to the operating environment shall have a prime record key whose value is greater than the highest prime record key value existing in the physical file when it was opened through that file connector and each subsequent record released to the operating environment through that file connector shall have a prime record key whose value is greater than the highest prime record key value written referencing this file connector. If the record is not in the sequence above, the execution of the WRITE statement is unsuccessful, and the I-O status value associated with file-name-1 is set based on your iscobol.file.status * setting.
35. If the access mode of the write file connector is random or dynamic, WRITE statements may release records to the operating environment through that connector in any order.
36. When the ALTERNATE RECORD KEY clause is specified in the file control entry associated with the write file connector, the value of the alternate record key may be nonunique only if the DUPLICATES phrase is specified for that data item. In this case the operating environment provides storage of records such that when records are accessed sequentially, the order of retrieval of those records is the order in which the operating environment actually writes the record into the physical file. If the DUPLICATES phrase is not specified and

the alternate key value is nonunique, the execution of the WRITE statement is unsuccessful, and the I-O status associated with file-name-1 is set based on your iscobol.file.status * setting.

37. The invalid key condition exists under the following circumstances. The comparison for equality for record keys is based on the collating sequence for the file according to the rules for a relation condition. Any record locks associated with the record being accessed are ignored in detection of these exceptions.
    A. When the write file connector is open for output or extend in the sequential access mode and the value of the prime record key is not greater than the value of the prime record key of the last record written through that file connector, the I-O status value associated with file-name-1 is set based on your iscobol.file.status * setting.
    B. When the value of the prime record key of the record to be written is equal to the value of the prime record key of any record existing in the file, the I-O status value associated with file-name-1 is set based on your iscobol.file.status * setting.
    C. When an alternate record key of the record to be written does not allow duplicates and the value of that alternate record key is equal to the value of the corresponding alternate record key of a record in the file, the I-O status value associated with file-name-1 is set based on your iscobol.file.status * setting.
    D. When the record that is to be released to the operating environment would reside beyond the externally defined boundaries of the physical file, the I-O status value associated with file-name-1 is set based on your iscobol.file.status * setting.

Format 4

38. A Format 4 WRITE statement writes an XML document to an I/O stream or add an XML element to the in-memory representation of the XML document.
39. WRITE without KEY IS takes the record specified, creates a new internal representation (if one does not exist already) and then writes that internal representation to the stream.

1. If the following conditions are met when an XML file is opened I-O:
    - an internal representation is created by a READ statement
    - the internal representation is modified by any combination of WRITE, REWRITE, or DELETE KEY statements
    - WRITE without the KEY IS clause is performed
2. The current contents of the record specified are ignored; instead, the internal representation as established by the READ and modified by the WRITE, REWRITE, or DELETE KEY statements, is written to the stream.
3. WRITE KEY IS PLAIN-TEXT, data-name-2 or literal-1 is a string value which contains plain text to be output to the XML stream.
4. WRITE KEY IS PLAIN-TEXT can be decoration such as "Content-type: text/xml."
5. No stream I/O is actually performed by the WRITE KEY IS statement.
6. WRITE KEY IS adds the node specified to the internal representation of the XML document immediately after the current position. If any containing nodes are required for the node specified, these are also created in the internal representation.
7. If ALL is specified, any contained nodes are also added to the internal representation of the XML document. Otherwise only the specific node and its attributes and data are added. No stream I/O is performed.
8. WRITE KEY IS PROCESSING-INSTRUCTION causes an XML processing-instruction node to be added immediately after the current position. When using sequential I/O, where your other WRITE statements have no usage of the KEY clause, a processing instruction is either written before or after the entire XML record, depending on order of execution. When using WRITE KEY IS PROCESSING-INSTRUCTION:
    - Data-name-2 is a defined level 78 or a simple data name.

- o Literal-1 is a string value which contains an XML-specific processing instruction. Upon output, this string is decorated with the XML delimiters '<? 'and '?>'.

9. WRITE KEY IS PLAIN-TEXT adds a plain text node immediately after the current position. When using sequential I/O, where your other WRITE statements have no usage of the KEY clause, plain text will either be written before or after the entire XML record, depending on order of execution.
10. If the internal representation of the XML document is modified through the use of WRITE KEY, REWRITE KEY, or DELETE KEY, and the internal representation is cleared with either a CLOSE or READ (no key) statement, then the CLOSE or READ statement returns a status of -10, indicating the operation succeeded but no write was done.

Examples

Format 1 - Print last lines to a page and advance 1 page

```
write report-line from page-subtotal-1-line after advancing 2 lines
write report-line from page-subtotal-2-line after advancing 2 lines
write report-line from page-total-line before advancing page
```

Format 2 - Write record validating invalid key

```
move 1234 to cust-code
move "Adam Smith" to cust-name
write cust-rec
      invalid key display message "Invalid Customer Key : " cust-code
      not invalid key display message "Customer saved!"
end-write
```

XML GENERATE

The XML GENERATE statement converts data to XML format.

General Format

```
XML GENERATE Xml-Stream FROM Xml-Data

    [ COUNT IN Counter ]

    [ ON EXCEPTION Imperative-Statement-1 ]

    [ NOT ON EXCEPTION Imperative-Statement-2 ]

[END-XML]
```

Syntax Rules

1. Xml-Stream must reference an elementary data item of category alphanumeric, an alphanumeric group item or an elementary data item of category national. If it references an alphanumeric group item, then it is treated as though it were an elementary data item of category alphanumeric.
2. Xml-Data can be a group or elementary data-item. It cannot be a function identifier or be reference modified, but it can be subscripted, and must not overlap with Xml-Stream or Counter.
3. Counter must be a numeric data item.

General Rules

1. If the COUNT IN phrase is specified, after execution of the XML GENERATE statement Counter contains the count of generated XML character positions.
2. An exception condition exists when an error occurs during generation of the XML document, for example if Xml-Stream is not large enough to contain the generated XML document. In this case the XML generation stops and the content of the receiver is undefined. If the COUNT IN phrase is specified, Counter contains the number of character positions that were generated, which can range from 0 to the length of Xml-Stream. If the ON EXCEPTION phrase is specified, control is transferred to Imperative-Statement-1, otherwise control is transferred to the end of the XML GENERATE statement.
3. If an exception condition does not occur during generation of the XML document, control is passed to Imperative-Statement-2, if specified, otherwise to the end of the XML GENERATE statement.
4. When the statement completes, two special registers are set:

    - XML-CODE contains the error type. Possible values are

0	Ok
1	Warning
10	Recoverable Error
100	Fatal Error

    - XML-ERRMSG contains the error message string, if an error occurred.

Examples

Generate an XML structure from a data structure

```
working-storage section.
 01 my-html.
    05 my-head  pic x(10) value "first".
    05 my-body.
       10 my-paragraph pic x(20) value "paragraph 1".
       10 my-table.
          15 my-row-1.
             20 my-col-1 pic x(20) value "col 1".
             20 my-col-2 pic x(20) value "col 2".
          15 my-row-2.
             20 my-col-1 pic x(20) value "col 1".
             20 my-col-2 pic x(20) value "col 2".
 01 the-xml  pic x any length.

 procedure division.
 main.
  xml generate the-xml from my-html
  end-xml
  display the-xml.
*> It will display the following:
*> <my-html><my-head>first</my-head><my-body><my-paragraph>paragraph 1</my-
paragraph><my-table><my-row-1><my-col-1>col 1</my-col-1><my-col-2>col 2</my-col-2></
my-row-1><my-row-2><my-col-1>col 1</my-col-1><my-col-2>col 2</my-col-2></my-row-2></
my-table></my-body></my-html>
```

XML PARSE

The XML PARSE statement parses an XML document into its individual pieces and passes each piece, one at a time, to a user-written processing procedure.

General Format

```
XML PARSE Xml-Stream PROCESSING PROCEDURE IS Procedure-Name-1
                                [ { THROUGH } Procedure-Name-2 ]
                                  { THRU    }

  [ ON EXCEPTION Imperative-Statement-1 ]

  [ NOT ON EXCEPTION Imperative-Statement-2 ]

[END-XML]
```

Syntax Rules

1. Xml-Stream must reference an elementary data item of category alphanumeric, an alphanumeric group item or an elementary data item of category national. If it references an alphanumeric group item, then it is treated as though it were an elementary data item of category alphanumeric.
2. Procedure-Name-1 is the first or only section or paragraph in the processing procedure.
3. Procedure-Name-2 is the last section or paragraph in the processing procedure.

General Rules

1. The processing procedure consists of the statements at which XML events are handled. The range of the processing procedure also includes all statements executed by CALL, EXIT, GO TO, GOBACK, and PERFORM statements in the range of the processing procedure.
2. The processing procedure must not directly execute an XML PARSE statement. However, if the processing procedure passes control to another program by using a CALL statement, the target program can execute the same or a different XML PARSE statement. A program executing on multiple threads can execute the same XML statement or different XML statements simultaneously.
3. An exception condition occurs when the XML parser detects an error in processing the XML document. The parser first signals an exception XML event by passing control to the processing procedure with special register XML-EVENT set to contain 'EXCEPTION'. If the XML processing procedure handles the exception XML event and sets XML-CODE to zero before returning control to the parser, the exception condition no longer exists. If no other unhandled exceptions occur prior to the termination of the parser, control is transferred to Imperative-Statement-2 of the NOT ON EXCEPTION phrase, if specified.
4. If an exception condition does not exist at termination of XML PARSE processing, control is transferred to Imperative-Statement-2 if the NOT ON EXCEPTION clause is specified. Otherwise control is transferred to the end of the XML PARSE statement.

Note: Unlike the IBM implementation of XML PARSE, isCOBOL doesn't return VERSION-INFORMATION and STANDALONE-DECLARATION events.

Examples

Parse a simple XML, display it and use some key data from it

```
 working-storage section.
 01 xml-document.
    02 pic x(36) value '<?xml version="1.0" encoding="utf-8"'.
    02 pic x(19) value ' standalone="yes"?>'.
    02 pic x(39) value '<!--This document is just an example-->'.
    02 pic x(10) value '<sandwich>'.
    02 pic x(35) value '  <bread type="baker&apos;s best"/>'.
    02 pic x(41) value '  <?spread please use real mayonnaise  ?>'.
    02 pic x(31) value '  <meat>Ham &amp; turkey</meat>'.
    02 pic x(40) value '  <filling>Cheese, lettuce, tomato, etc.'.
    02 pic x(10) value '</filling>'.
    02 pic x(35) value '  <![CDATA[We should add a <relish>'.
    02 pic x(22) value ' element in future!]]>'.
    02 pic x(31) value '  <listprice>$4.99 </listprice>'.
    02 pic x(27) value '  <discount>0.10</discount>'.
    02 pic x(15) value '</sandwich>'.

 01 xml-document-length computational pic 999.

 01 my-msg                    pic x(30).
 01 current-element           pic x(30).
 01 xfr-ed                    pic x(9) justified.
 01 xfr-ed-1 redefines xfr-ed pic 999999.99.
 01 list-price computational  pic 9v99 value 0.
 01 discount computational    pic 9v99 value 0.
 01 display-price             pic $$9.99.
 01 prom-price                pic $$9.99.

procedure division.
 main.
   display  xml-document
   xml parse xml-document processing procedure xml-handler
       on exception     display 'XML document error ' xml-code x'0d0a' xml-errmsg
       not on exception display 'XML document successfully parsed'
   end-xml
   move list-price to display-price
   compute prom-price = list-price * (1 - discount)
   display 'Information from XML ' x'0d0a'
           '  Sandwich list price: ' display-price x'0d0a'
           '  Promotional price:   ' prom-price
   end-display
   goback.
xml-handler.
   evaluate xml-event
```

```
    when 'DOCUMENT-TYPE-DECLARATION ' display 'Doc type decl: ' XML-TEXT
    when 'START-OF-ELEMENT'
      display 'Start element tag: <' XML-TEXT '>'
      move XML-TEXT to current-element
    when 'CONTENT-CHARACTERS'
      display 'Content characters: <' XML-TEXT '>'
      evaluate current-element
        when 'listprice' compute list-price = function numval-c(XML-TEXT)
        when 'discount'
          move XML-TEXT to xfr-ed
          move xfr-ed-1 to discount
      end-evaluate
    when 'END-OF-ELEMENT'
      display 'End element tag: <' XML-TEXT '>'
      move spaces to current-element
    when 'START-OF-DOCUMENT'
      compute xml-document-length = function length(XML-TEXT)
      display 'Start of document: length=' xml-document-length
          ' characters.'
    when 'END-OF-DOCUMENT' display 'End of document.'
    when 'VERSION-INFORMATION' display 'Version: <' XML-TEXT '>'
    when 'ENCODING-DECLARATION' display 'Encoding: <' XML-TEXT '>'
    when 'STANDALONE-DECLARATION' display 'Standalone: <' XML-TEXT '>'
    when 'ATTRIBUTE-NAME' display 'Attribute name: <' XML-TEXT '>'
    when 'ATTRIBUTE-CHARACTERS' display 'Attribute value characters: <' XML-TEXT '>'
    when 'ATTRIBUTE-CHARACTER' display 'Attribute value character: <' XML-TEXT '>'
    when 'START-OF-CDATA-SECTION' display 'Start of CData: <' XML-TEXT '>'
    when 'END-OF-CDATA-SECTION' display 'End of CData: <' XML-TEXT '>'
    when 'CONTENT-CHARACTER' display 'Content character: <' XML-TEXT '>'
    when 'PROCESSING-INSTRUCTION-TARGET' display 'PI target: <' XML-TEXT '>'
    when 'PROCESSING-INSTRUCTION-DATA' display 'PI data: <' XML-TEXT '>'
    when 'COMMENT' display 'Comment: <' XML-TEXT '>'
    when 'EXCEPTION'
      compute xml-document-length =
                function length (XML-TEXT)
     move  XML-TEXT (xml-document-length - 5:10) to  my-msg
     display 'Exception ' XML-CODE ' at offset '
         xml-document-length ':' xml-errmsg ':' my-msg
    when other display 'Unexpected XML event: ' XML-EVENT '.'
  end-evaluate.
```

YIELD

General Format

```
YIELD
```

General rules

The YIELD statement is used in conjunction with the STOP THREAD statement. STOP THREAD requires YIELD to work correctly, otherwise, the thread will NOT be released.

For Example:

```
        perform thread show-time handle t1.
        ...
        STOP THREAD t1.
        ...

    show-time.
        perform until exit
           accept w-time from time
           display w-time line 2 pos 70
           YIELD
        end-perform.
```

Chapter 8

Compiler-Directing Statements

A compiler-directing statement begins with a compiler-directing verb that causes the compiler to perform a specific operation during compilation. The following section covers the verbs available in isCOBOL.

COPY

Format 1

```
COPY library-name [ {IN} path-name ] [ SUPPRESS PRINTING]
                  {OF}

                [ REPLACING { { old-text BY new-text                } } ... ] .
                            { { {LEADING } literal-1 BY {literal-2} } }
                            { { {TRAILING}             {SPACE    } } }
                            { {                        {SPACES   } } }
```

Format 2

```
COPY RESOURCE resource-name  [ {IN} path-name ]
                               {OF}
```

Syntax rules

1. The COPY statement must be terminated by a period.
2. *Library-name, path-name* and *resource-name* must be a nonnumeric literals or user-defined words.
3. COPY statements may be nested in other COPY libraries.
4. Any one of the COPY statements in this structure can include the REPLACING phrase.
5. *Old-text* and *new-text* may be any of the following:
   A. A series of text words placed between "==" delimiters.
   B. A numeric or nonnumeric literal.
   C. A data name, including qualifiers, subscripts and reference modification.
   D. Any single text word.
6. *Resource-name* and *path-name* identify a resource file to be included in the resulting object file.

General rules

Format 1

1. Library-name and path-name identify a source file to be included at the location of the COPY statement. If library-name is delimited by quotes, the source copy file is searched using the case specified by the statement. If library-name is not delimited by quotes, the source copy file is searched first using the case specified by the statement, if it's not found, then it's searched in all upper-case. For example:

```
COPY "MyFile.Cpy"
```

   will search only for *MyFile.Cpy*.

```
COPY MyFile.Cpy
```

   will search first for *MyFile.Cpy* and then for *MYFILE.CPY*.

   The file extension is converted to upper-case along with the file name only if specified in the source code. File extensions specified through -ce=Ext1... compiler option are left unchanged. For example, having

```
COPY MyFile
```

   and compiling with *-ce=Cpy* will search first for *MyFile.Cpy* and then for *MYFILE.Cpy*.

2. If the word SUPPRESS appears after library-name and path-name, then the program listing file will not include the contents of the library file or any other library files that may be nested within.
3. The text of the library file is copied unchanged into the source program unless the REPLACING option is used. If the REPLACING option is used, then elements of the library file that match old-text or literal-1 are replaced by new-text or literal-2.
4. 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 old-text or literal-1 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. Old-text matches the library text only if the ordered sequence of text words that forms old-text is equal, character for character, to the ordered sequence of library text words. When the LEADING phrase is specified, literal-1 matches the library text only if the contiguous sequence of characters that forms partial-word-1 is equal, character for character, to the leading characters of the library text word. When the TRAILING phrase is specified, literal-1 matches the library text only if the contiguous sequence of characters that forms literal-1 is equal, character for character, to the trailing characters of the library text word.
   C. The following rules apply for the purpose of matching:
      i. Each occurrence of a separator comma, semicolon, or space in old-text 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.
      ii. 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.
      iii. A compiler directive line in source text shall be considered a single text word that does not match any text word within old-text or literal-1.
      iv. Comments or blank lines occurring in the source text and in old-text are ignored.

v. If no match occurs, the comparison is repeated with each next successive old-text or literal-1, if any, in the REPLACING phrase until either a match is found or there is no next successive REPLACING operand.

vi. 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 old-text or literal-1 specified in the REPLACING phrase. When a match occurs between old-text and the library text, the corresponding new-text 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 literal-2 or deleted when literal-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 old-text or literal-1 specified in the REPLACING phrase.

vii. 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.

D. 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.

E. Comments or blank lines appearing in new-text are copied into the source text unchanged whenever new-text 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 old-text.

F. The REPLACING phrase can be used to replace substrings.
To make use of this, delimit the substring that will be replaced in the COPY library with quotes. Then use the standard COPY syntax to replace the quoted substring by another substring. The resulting sequence of characters is re-evaluated by the compiler to make a new string.

For example, suppose you have a COPY library called "MYLIB" that contains the following:

```
77 THE-'DUMMY'-DATA-ITEM PIC X(10).
```

and you used this COPY statement:

```
COPY "MYLIB" REPLACING =='DUMMY'== BY ==REAL==.
```

Then the text of "MYLIB" is effectively treated as:

```
77 THE-REAL-DATA-ITEM PIC X(10).
```

5. When the -sevc option is used, you may use operating system environment variables as well as configuration properties in the OF phrase of a COPY statement. To reference an environment variable or a configuration property, place a $ in front of it. For example, if you assign ENVPATH to C:\Develop\Copybooks, then the statement

```
COPY "CUSTDATA" OF "$ENVPATH".
```

would use the file C:\Develop\Copybooks\CUSTDATA.

To assign ENVPATH in the operating system environment, use *SET ENVPATH=C:\Develop\Copybooks* on Windows and *export ENVPATH=C:\Develop\Copybooks* on Linux/Unix. To assign ENVPATH as a property in the Compiler configuration, use *iscobol.envpath=C:\\Develop\\Copybooks*.

You may use multiple environment variables by preceding each one with a $ symbol. Symbol names may contain alphanumeric characters, hyphens, underscores, and dollar signs. If the symbol name is not found in the environment, it is left unchanged (including the initial $ symbol). Symbols are not processed recursively; if the value of a symbol contains a $, the dollars sign is used literally in the final file name.

Format 2

6. The effect of a COPY RESOURCE statement is to add *resource-name* to a list of resources that the compiler embeds into the resulting class file. Conventionally, COPY RESOURCE statements are placed either in Working-Storage or at the end of the program, but any location is acceptable.
7. The resource is compressed before being added to the class file.
8. After *resource-name* has been included in the compiled class file, it can be accessed without case sensitivity by COBOL routines.
9. On case sensitive systems like Linux, having multiple COPY RESOURCE statements that include files with the same name but different case, the compiler will include the last of these files.

EJECT

General Format

```
EJECT
```

General Rules

1. The EJECT statement specifies that the next source statement is to be printed at the top of the next page.
2. The EJECT statement must be the only statement on the line. IBM DOS/VS COBOL requires these words to appear in Area B and can be terminated with a separator period.
3. The EJECT statement has no effect on the compilation of the source unit itself.

PROCESS

General Format

```
{CBL       } options
{PROCESS   }
```

Syntax Rules

1. CBL and PROCESS are synonym.

General Rules

1. The -cv option is required in order to compile this syntax.

2. PROCESS must be located before the IDENTIFICATION DIVISION if present, otherwise before PROGRAM-ID.
3. *Options* must appear in the same line.
4. Multiple PROCESS statements can be used.
5. The PROCESS statement is ignored by the Compiler. The syntax is compiled only for compatibility purposes.

REPLACE

General Format

Format 1

```
REPLACE  [ ALSO ] { { old-text BY new-text                } } ... ] .
                  { { {LEADING } literal-1 BY {literal-2} } }
                  { { {TRAILING}              {SPACE    } } }
                  { {                         {SPACES   } } }
```

Format 2

```
REPLACE [ LAST ] OFF
```

Syntax rules

1. *Old-text* and *new-text* may be any of the following:
   A. A series of text words placed between "==" delimiters.
   B. A numeric or nonnumeric literal.
   C. A data name, including qualifiers, subscripts and reference modification.
   D. Any single text word.

General rules

1. *Old-text:*
   A. Must contain one or more text words. Character-strings can be continued in accordance with normal source code rules.
   B. Can consist solely of a separator comma or a separator semicolon
2. *new-text:*
   A. Can contain zero, one, or more text words. Character strings can be continued in accordance with normal source code rules.
3. *Old-text* and *new-text* can contain any text words that can be written in source text.
4. The compiler processes REPLACE statements in source text after the processing of any COPY statements. COPY must be processed first, to assemble complete source text. Then REPLACE can be used to modify that source text, performing simple string substitution.
5. The text produced as a result of the processing of a REPLACE statement must not contain a REPLACE statement.
6. A given occurrence of the REPLACE statement is in effect from the point at which it is specified until the next occurrence of the REPLACE statement. The new REPLACE statement supersedes the text-matching established by the previous REPLACE statement unless the ALSO phrase is specified.
7. A format 2 REPLACE suspends all replacements currently in effect.

SKIP

The SKIP statements specify blank lines that the compiler should add when printing the source listing. SKIP statements have no effect on the compilation of the source text itself

General Format

```
SKIP1
SKIP2
SKIP3
```

General Rules

1. The SKIP statement must be the only statement on the line. IBM DOS/VS COBOL requires these words to appear in Area B and can be terminated with a separator period.
2. The SKIP statement has no effect on the compilation of the source unit itself.
3. SKIP1 skips one line before the next line
4. SKIP2 skips two lines before the next line
5. SKIP3 skips three lines before the next line

USE

Format 1

```
USE [GLOBAL] AFTER STANDARD {EXCEPTION} PROCEDURE ON { { File-1 } ... }
                            {ERROR    }              { INPUT          }
                                                     { OUTPUT         }
                                                     { I-O            }
                                                     { EXTEND         }
                                                     { TRANSACTION    }
```

Format 2

```
USE AT PROGRAM { START }
               { END   }
```

Format 3

```
USE FOR DEBUGGING ON { ProcedureName    }
                     { ALL PROCEDURES   }
```

Syntax rules

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-1* shall not be a sort or a merge file.
3. Within a declarative procedure, there shall be no reference to any nondeclarative procedures except in a RESUME statement.
4. Procedure-names within a declarative section may be referenced in a different declarative section or in a

nondeclarative procedure only with a PERFORM statement.

5. The files implicitly or explicitly referenced in the USE statement need not all have the same organization or access.
6. The words ERROR and EXCEPTION are synonymous and may be used interchangeably.
7. The INPUT, OUTPUT, I-O, and EXTEND phrases may each be specified only once in the declaratives portion of a given procedure division.
8. The same file-name shall not appear in more than one USE AFTER EXCEPTION statement within the same procedure division.
9. The Format 3 USE statement is supported only under the -cv compiler option.

General rules

1. The USE statement is never executed; it merely defines the conditions calling for the execution of the USE procedures.
2. During the execution of a USE procedure, if a statement raises an exception condition that would cause the execution of a USE procedure that had previously been activated and had not yet returned control to the activating entity, an overflow exception condition is set to exist.
3. A declarative is selected for execution by analyzing the USE statements in the source element in the order in which they are specified. The first declarative that satisfies the selection criteria is executed and no other declaratives are executed.
4. 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.

Format 1

5. Within a given procedure division, a USE statement specifying *File-1* takes precedence over any USE statements specifying an INPUT, OUTPUT, I-O, or EXTEND phrase.
6. 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.

Format 2

7. A Format 2 USE statement creates a START or END procedure for the program. No more than one START and one END procedure is allowed for each program.
   A. A START procedure executes immediately before the first normal COBOL statement in the Procedure Division when the program is in its initial state. The START procedure executes only once regardless of the number of times the program is entered, until the program is cancelled from memory. A START procedure executes regardless of which entry point is used to start the program when a program contains multiple entry points.
   B. An END procedure executes immediately before the program is placed into its initial state or it is about to leave memory, providing the program has been entered at least once.
8. START and END procedures should not reference data passed to the program through the Linkage Section.

Format 3

9. A Format 3 USE statement identifies the items in the source program that are to be monitored by the associated debugging procedure.
10. The DEBUG procedure is executed only if the program includes WITH DEBUGGING MODE in the Source-Computer paragraph and if the iscobol.use_for_debugging (boolean) * configuration property is set to true in the configuration.
11. It's possible to specify a DEBUG procedure for specific paragraphs and sections as well as a generic DEBUG procedure for all the paragraphs and sections in the program. Use the syntax "ON ProcedureName", where ProcedureName is the name of a paragraph or section, to attach a DEBUG procedure to a specific paragraph or section. Use "ON ALL PROCEDURES" to attach the same DEBUG procedure to all the paragraphs and sections in the program.
12. Within a DEBUG procedure the special register DEBUG-NAME is set to the name of the current paragraph or section.

Chapter 9

Embedded SQL (ESQL)

Embedded SQL is a method of combining the computing power of a high-level language and the database manipulation capabilities of SQL. It allows you to execute SQL statements from an application program.

All SQL statements need to be enclosed within `EXEC SQL` and `END-EXEC`. You can place the SQL statements anywhere within a program, with the restriction that variables and constants must be declared in file, working-storage or linkage section, while executable SQL statements must be placed in procedure division.

Identifiers

The database object name is known as its identifier. Servers, databases, and database objects such as tables, views, columns, indexes, triggers, procedures, constraints, rules, and so on can have identifiers. An object identifier is created when the object is defined. The identifier is then used to reference the object.

Note: The Compiler automatically translates identifiers to upper-case replacing hyphens with underscores unless the identifiers are enclosed between double quotes. In addition, the Compiler translates double quotes to single quotes unless the -csqq option is used.

The following statement, for example:

```
SELECT column-1, column-2 FROM table-1 WHERE column-1 = "x"
```

becomes:

```
SELECT COLUMN_1, COLUMN_2 FROM TABLE_1 WHERE COLUMN_1 = 'x'
```

The following statement, instead, is left unaltered, assuming that the program is compiled with -csqq:

```
SELECT "column-1", "column-2" FROM "table-1" WHERE "column-1" = 'x'
```

Host Variables

Host variables allow the host program and the database to communicate. A host variable is any COBOL variable or constant, declared in file, working-storage or linkage section.

A host variable reference must be prefixed with a colon ":" in SQL statements.

Code example

The content of column1 is sotred in the host variable named item1.

```
...

working-storage section.

...

exec sql include SQLCA end-exec.
77 item1 pic 9(3).
...

procedure division.

...

exec sql select column1 into :item1 from table1 where column1 = 1 end-exec.
...
```

Note: alphanumeric data items with picture X(n) are internally translated to their unicode representation according to the current locale when they're used as host variables. This rule may cause unexpected data translation if the content of the data item is a non representable character.

Update of host variables value

Host variables are not altered if the ESQL statement fails due to an SQL error.

If the statement fails, the host variables preserve the value they had before the statement was executed.

How to specify the parent data item

The parent data item of a host variable can be referenced using a separator dot or the OF keyword. The IN keyword cannot be used as the compiler may confuse it with a stored procedure parameter type.

Given the following group items:

```
01 group-1.
   03 item-1 pic x(30).
01 group-2.
   03 item-1 pic 9(9).
```

In order to avoid ambiguous identifiers, you can reference item-1 of group-1 in these ways

- :group1.item-1
- :item-1 of group1

Group Items as Host Variables

isCOBOL allows the use of group items in embedded SQL statements. Group items with elementary items (containing only one level) can be used as host variables. The host group items (also referred to as host structures) can be referenced in the INTO clause of a SELECT or a FETCH statement, and in the VALUES list of an INSERT statement. When a group item is used as a host variable, only the group name is used in the SQL statement. For example, given the following declaration

```
       01  meeting-time.
           05 hour    pic 9(2).
           05 minute  pic 9(2).
```

the following statement is valid:

```
           exec sql
              select mhour, mminute
                into :meeting-time
                from meetings
               where meeting_id = 1
           end-exec.
```

The order that the members are declared in the group item must match the order that the associated columns occur in the SQL statement, or in the database table if the column list in the INSERT statement is omitted. Using a group item as a host variable has the semantics of substituting the group item with elementary items. In the above example, it would mean substituting :meeting-time with :meeting-time.hour, :meeting-time.minute.

Storing binary data in Host Variables

After the host variable declaration, the following syntax can be used:

```
EXEC SQL VAR Host-Variable IS Type END-EXEC
```

Where

- *Host-Variable* is a host variable.
- *Type* can be one of the following keywords:
  - BINARY
  - LONG RAW
  - RAW

  Any other value is treated as a comment and doesn't have effect

With this syntax a specific variable is marked as storage for binary data. The runtime is responsible to manage the variable content according to the current database specifications.

Code example

The content of the column t1_field_2 of type RAW in a Oracle table is shown in hex format after reading

```
...

working-storage section.

...

exec sql include SQLCA end-exec.
77 wrk-ascii pic x(128).
exec sql var wrk-ascii is raw end-exec.
77 wrk-hex   pic x(256).

...

procedure division.

...

exec sql
     select t1_field_2 into :wrk-ascii from table1 where t1_pk = 1
end-exec.
call "ASCII2HEX" using wrk-ascii, wrk-hex.
display "t1_field_2 = " wrk-hex.
...
```

Mapping a VARCHAR field to a COBOL group data item

Explicit definition

Database VARCHAR fields can be mapped to a COBOL group data item with the following structure:

```
01 ITEM-NAME.
   49 ITEM-LEN  PIC S9(4) USAGE BINARY.
   49 ITEM-TEXT PIC X(n).
```

Note that the mapping occurs only if the parent item has level 01 and the two children items have level 49. There are no conditions on the name of the data items, instead.

The ITEM-TEXT data item can be any size. The USAGE of the ITEM-LEN data item can be BINARY, COMP, COMP-4 or COMP-5.

When this kind of host variable is used as input (e.g. in an INSERT statement), the ITEM-TEXT data item specifies the text to be stored in the VARCHAR field and the ITEM-LEN data item specifies the length of the inserted value. If the value of ITEM-LEN is less than the length of the value in ITEM-TEXT, a truncation occurs. If the value of ITEM-LEN is greater than the length of the value in ITEM-TEXT, the exceeding part is filled with spaces.

When this kind of host variable is used as output (e.g. in a SELECT statement), the ITEM-TEXT data item receives the text stored in the VARCHAR field and the ITEM-LEN data item receives the length of the retrieved value.

Note - trailing spaces in the value may be preserved or not depending on the iscobol.jdbc.kept_spaces configuration setting. The LEN data item will be adjusted accordingly by the runtime.

Implicit definition

Database VARCHAR fields can be mapped to a COBOL data item with a Format 3 PICTURE clause:

```
01 ITEM-NAME PIC X(n) VARYING.
```

ITEM-NAME data item can be any name and size.

This kind of item is internally expanded by the Compiler as follows:

```
01  ITEM-NAME.
    03  ITEM-NAME-arr PIC X(n).
    03  ITEM-NAME-len PIC S9(4) COMP-5.
```

When this kind of host variable is used as input (e.g. in an INSERT statement), the data item with "-arr" suffix specifies the text to be stored in the VARCHAR field and the data item with "-len" suffix specifies the length of the inserted value. If the value of the len item is less than the length of the value in the arr item, a truncation occurs. If the value of the len item is greater than the length of the value in the arr item, the exceeding part is filled with spaces.

When this kind of host variable is used as output (e.g. in a SELECT statement), the data item with "-arr" suffix receives the text stored in the VARCHAR field and the data item with "-len" suffix receives the length of the retrieved value.

Note - trailing spaces in the value may be preserved or not depending on the iscobol.jdbc.kept_spaces configuration setting. The data item with "-len" suffix will be adjusted accordingly by the runtime.

Code example

Insert a value using a VARYING data item and read it back using a group data item:

```
      *Table structure on the database:
      *      create table tbl (id numeric(1), vc varchar(10))
      *
       program-id. vchar.
       working-storage section.
       ...
       01 itm pic x(10) varying.
       01 gr.
          49 ln pic s9(4) usage binary.
          49 txt pic x(10).
       ...
       procedure division.
       main.
           exec sql
                connect :usr identified by :pwd using :dns
           end-exec.
           move "test" to itm-arr.
           move 4      to itm-len.
           exec sql
                insert into tbl(id, vc)
                       values (1, :itm)
           end-exec.
           exec sql
                select vc into :gr
                          from tbl
                          where id = 1
           end-exec.
           display "'" txt(1:ln) "'".   | it will display 'test'
```

Mapping Large Objects

isCOBOL proprietary syntax

Database large object (LOB) fields can be managed as follows:

For BLOB fields, use the ESQL$BLOB library routine.

For CLOB fields, use an host variable with picture X ANY LENGTH if the CLOB length is unknown or use an host variable with picture X(n) where n matches the length in bytes of the CLOB column.

Example.

```
       PROGRAM-ID. readwritelob.

       WORKING-STORAGE SECTION.
       copy "SQLCA".
       copy "iscobol.def".

       77 W-KEY   pic 9(4).
       77 W-DATA  pic x(30).
       77 W-CLOB  pic x any length.
       77 W-BLOB  HANDLE.

       PROCEDURE DIVISION.
       Main.
           CALL "ESQL$BLOB" USING GET-BLOB-FROM-FILE, W-BLOB,"img1.bmp".

           EXEC SQL
                CONNECT
           END-EXEC

           EXEC SQL
                DROP TABLE IS_TABLE
           END-EXEC

           EXEC SQL
                CREATE TABLE IS_TABLE
                             (IS_KEY INT NOT NULL,
                              IS_DATA CHAR(6),
                              IS_CLOB CLOB,
                              IS_BLOB BLOB)
           END-EXEC

           EXEC SQL
                ALTER TABLE IS_TABLE ADD PRIMARY KEY (IS_KEY)
           END-EXEC

           MOVE "CLOB data" TO W-CLOB.

           EXEC SQL INSERT INTO IS_TABLE VALUES (1, 'row1',
                                                 :W-CLOB,
                                                 :W-BLOB)
           END-EXEC

           CALL "ESQL$BLOB" USING FREE-BLOB-HANDLE, W-BLOB.

           EXEC SQL
                SELECT * INTO :W-KEY, :W-DATA, :W-CLOB, :W-BLOB
                             FROM IS_TABLE
                             WHERE IS_KEY = 1
           END-EXEC

           CALL "ESQL$BLOB" USING PUT-BLOB-INTO-FILE, W-BLOB, "blob.bmp".

           EXEC SQL
                DISCONNECT
           END-EXEC

           GOBACK.
```

IBM DB2 compatible syntax

Database large object (LOB) fields can be mapped to a COBOL data item with a SQL TYPE USAGE clause:

```
01 variable-name USAGE IS SQL TYPE IS { BLOB   } ( length {K} ).
                                      { CLOB   }          {M}
                                      { DBCLOB }          {G}
```

For BLOB and CLOB, length must be between 1 and 2,147,483,647.

For DBCLOB, length must be between 1 and 1,073,741,823.

SQL TYPE IS, BLOB, CLOB, DBCLOB, K, M, G can be in either uppercase, lowercase, or mixed.

Initialization within the LOB declaration is not permitted.

The host variable name prefixes LENGTH and DATA in the precompiler generated code.

BLOB Example:

Declaring:

```
01 MY-BLOB USAGE IS SQL TYPE IS BLOB(2M).
```

Results in the generation of the following structure:

```
01 MY-BLOB.
   49 MY-BLOB-LENGTH PIC S9(9) COMP-5.
   49 MY-BLOB-DATA PIC X(2097152).
```

CLOB Example:

Declaring:

```
01 MY-CLOB USAGE IS SQL TYPE IS CLOB(125M).
```

Results in the generation of the following structure:

```
01 MY-CLOB.
  49 MY-CLOB-LENGTH PIC S9(9) COMP-5.
  49 MY-CLOB-DATA PIC X(131072000).
```

DBCLOB Example:

Declaring:

```
01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(30000).
```

Results in the generation of the following structure:

```
01 MY-DBCLOB.
   49 MY-DBCLOB-LENGTH PIC S9(9) COMP-5.
   49 MY-DBCLOB-DATA PIC G(30000) DISPLAY-1.
```

Although the syntax is supported in compatibility with the IBM DB2 preprocessor, it can potentially work on every database as long as the LOB data type specified by the SQL TYPE clause is supported by the database.

SQL and COBOL data types

The table below shows a list of the most common SQL data types and the corresponding COBOL data type. You should define the host variables of your COBOL program according to this table.

SQL	COBOL
BLOB	PIC X(n), USAGE IS SQL TYPE BLOB or ESQL$BLOB routine
CHAR (n)	PIC X(n)
CLOB	PIC X(n) or USAGE IS SQL TYPE CLOB
DATE / TIME	PIC X(n) and the proper date conversion function, that varies depending on the database
DECIMAL(n,d)	PIC S9(n-d)v9(d)
NCHAR(n)	PIC N(n)
NCLOB	PIC N(n)
NUMERIC(n)	PIC S9(n)
NVARCHAR(n)	PIC N(n)
VARCHAR (n)	PIC X(n) or group item as described in Mapping a VARCHAR field to a COBOL group data item

Indicator Variables

You can associate any host variable with an optional indicator variable. Each time the host variable is used in a SQL statement, a result code is stored in its associated indicator variable. Thus, indicator variables let you monitor host variables.

Indicator variables must be declared as a 2-byte integer (e.g. PIC S9(4) COMP).

If they're not explicitly defined in the program's Working-Storage, the compiler defines them implicitly and returns the warning number 265.

Their references must be prefixed with a colon ":" in SQL statements and must follow the host variable they refer to. There can be no space, one space or more spaces between the host variable and its indicator variable. The indicator variable can even appear on a separate line. The compiler uses the comma character to distinguish between host variables and indicator variables, for example:

`:var1, :var2`	var1 and var2 are considered as two separate host variables
`:var1 :var2`	var1 is considered host variable while var2 is considered indicator variable

To improve readability, you can precede any indicator variable with the optional keyword INDICATOR. You must still prefix the indicator variable with a colon. The correct syntax is:

```
:host_var INDICATOR :indicator_var
```

which is equivalent to

```
:host_var:indicator_var
```

Possible values for indicator variables are:

Input	-1	The value of the host variable is ignored and NULL will be used, instead.
	>=0	The whole value of the host variable will be used, no matter the value of the indicator variable.
Output	-2	The column value does not fit the host variable, but the length of the column value cannot be determined.
	-1	The column value is NULL. The program may not use the value of the host variable.
	0	The column value fits the host variable.
	>0	The column value does not fit the host variable. The indicator variable value is the length, in characters for national items, of the column value. When the data truncation occurs on a output parameter of a stored procedure, the behavior is influenced by the iscobol.esql.indicator_trunc_on_call (boolean) configuration property.

Code example

The content of column1 is stored in the host variable named item1. The indicator variable ind1 is also set.

```
...

working-storage section.

...

exec sql include SQLCA end-exec.
77 item1 pic 9(3).
77 ind1  pic s9(4) comp.
...

procedure division.

...

exec sql select column1 into :item1:ind1 from table1 where column1 = 1 end-exec.
...
```

Dynamic SQL

While embedded SQL is fine for fixed applications, sometimes it is important for a program to dynamically create entire SQL statements. With dynamic SQL, a statement stored in a string variable can be issued. PREPARE turns a character string into a SQL statement, and EXECUTE executes that statement.

PREPARE and EXECUTE may be combined into one statement using the IMMEDIATE clause of the EXECUTE statement.

Code example

```
...

working-storage section.

...

exec sql include SQLCA end-exec.
77 cmd usage handle.
...

procedure division.

...

exec sql prepare :cmd from "DELETE FROM TABLE1" end-exec.
exec sql execute :cmd end-exec.
...
```

SQLCA

SQLCA (SQL Communications Area) is used to detect errors and status changes in your program. This structure contains components that are filled in by the database at runtime after every executable SQL statement.

To use embedded SQL statements you must include SQLCA in working-storage section using the INCLUDE statement:

```
...
WORKING-STORAGE SECTION.
...
EXEC SQL INCLUDE SQLCA END-EXEC
...
```

isCOBOL supports three formats of the SQLCA:

Format 1

```
01  SQLCA.
    05  SQLCAID                PIC X(8).
    05  SQLCABC                PIC S9(9) COMP-5.
    05  SQLCODE                PIC S9(9) COMP-5.
    05  SQLERRM.
        49  SQLERRML           PIC S9(4) COMP-5.
        49  SQLERRMC           PIC X(254).
    05  SQLERRP                PIC X(8).
    05  SQLERRD OCCURS 6 TIMES PIC S9(9) COMP-5.
    05  SQLWARN.
        10 SQLWARN0            PIC X(1).
        10 SQLWARN1            PIC X(1).
        10 SQLWARN2            PIC X(1).
        10 SQLWARN3            PIC X(1).
        10 SQLWARN4            PIC X(1).
        10 SQLWARN5            PIC X(1).
        10 SQLWARN6            PIC X(1).
        10 SQLWARN7            PIC X(1).
    05  SQLSTATE               PIC X(5).
    05  SQLEXT                 PIC S9(5) COMP-3 VALUE 1.
```

Format 2

```
01  SQLCA GLOBAL.
    05  SQLCAID                PIC X(8).
    05  SQLCABC                PIC S9(9) COMP-5.
    05  SQLCODE                PIC S9(9) COMP-5.
    05  SQLERRM.
        49 SQLERRML            PIC S9(4) COMP-5.
        49 SQLERRMC            PIC X(70).
    05  SQLERRP                PIC X(8).
    05  SQLERRD OCCURS 6 TIMES
                               PIC S9(9) COMP-5.
    05  SQLWARN.
        10 SQLWARN0            PIC X(1).
        10 SQLWARN1            PIC X(1).
        10 SQLWARN2            PIC X(1).
        10 SQLWARN3            PIC X(1).
        10 SQLWARN4            PIC X(1).
        10 SQLWARN5            PIC X(1).
        10 SQLWARN6            PIC X(1).
        10 SQLWARN7            PIC X(1).
    05  SQLEXT                 PIC X(8).
```

Format 3

```
01 SQLCA.
   05 SQLCAID       PIC X(8).
   05 SQLCABC       PIC S9(9) BINARY.
   05 SQLCODE       PIC S9(9) BINARY.
   05 SQLERRM.
      49 SQLERRML   PIC S9(4) BINARY.
      49 SQLERRMC   PIC X(70).
   05 SQLERRP       PIC X(8).
   05 SQLERRD       OCCURS 6 TIMES
                    PIC S9(9) BINARY.
   05 SQLWARN.
      10 SQLWARN0   PIC X(1).
      10 SQLWARN1   PIC X(1).
      10 SQLWARN2   PIC X(1).
      10 SQLWARN3   PIC X(1).
      10 SQLWARN4   PIC X(1).
      10 SQLWARN5   PIC X(1).
      10 SQLWARN6   PIC X(1).
      10 SQLWARN7   PIC X(1).
      10 SQLWARN8   PIC X(1).
      10 SQLWARN9   PIC X(1).
      10 SQLWARNA   PIC X(1).
   05 SQLSTATE      PIC X(5).
```

The framework automatically chooses the proper format with the following criteria:

If the SQLCA length is 320 bytes, then Format 1 is used.

If the SQLCA length is 136 bytes, then Format 2 is used.

If the SQLCA length is 136 bytes and the -csdb2 compiler option is used, then Format 3 is used.

If none of the above conditions is true, then an *Invalid SLQCA* runtime error occurs.

SQLCA fields

SQLCAID	Not used
SQLCABC	Not used
SQLCODE	Contains the SQL return code. A value of zero means success, although one or more SQLWARN indicators may be set. The value 100 means "record not found" and it's usually returned only by SELECT and FETCH statements. Set the iscobol.esql.value_sqlcode_on_no_data configuration property in order to have this value (or another value of your choice) returned also by other statements. Any other value is an error condition. The error code is database dependent.
SQLERRML	Contains the length of sqlerrmc. 0 means that the value of sqlerrmc is not relevant.
SQLERRMC	Contains the description of the error condition.
SQLERRP	Not used
SQLERRD(1)	Not used
SQLERRD(2)	Not used
SQLERRD(3)	Contains the number of rows affected by the SQL statement
SQLERRD(4)	Not used
SQLERRD(5)	Not used
SQLERRD(6)	Not used
SQLWARN	A set of warning indicators, each containing a blank or W. The meaning of each indicator is database dependent.
SQLSTATE	SQLSTATE status codes consist of a 2-character class code immediately followed by a 3-character subclass code. Aside from class code 00 ("successful completion",) the class code denotes a category of exceptions. And, aside from subclass code 000 ("not applicable",) the subclass code denotes a specific exception within that category. The meaning of codes is database dependent. The runtime sets this field by invoking the getSQLState method of the Java SQLException behind the error condition.
SQLEXT	Not used

Record not found / No more records

SQLCODE is set to 100 in these two conditions:

- no more records available, end of result set, on FETCH statement
- no records found on a singleton SELECT statement

SQLERRD(3) can be checked to know how many records have been added or altered after UDDATE, DELETE and INSERT.

Therefore, in order to know if the WHERE condition of an UPDATE or DELETE found some records or not, SQLERRD(3) must be checked.

Customization through an handler class

If the iscobol.esql.sqlca_handler * configuration property specifies a valid class, then the sqlcaDecoder() method of that class is invoked to retrieve the values for SQLCODE, SQLERRMC and SQLSTATE. The runtime passes the current values of these fields along with the underlying java.sql.Exception (if any) to the class. The following snippet shows a prototype of a valid class

```
import com.iscobol.types.CobolVar;
import com.iscobol.rts.EsqlSqlcaHandler;
import java.sql.SQLException;

public class MySqlcaHandler implements EsqlSqlcaHandler {

   public void sqlcaDecoder(SQLException ex, CobolVar sqlcode,
                                    CobolVar sqlstate, CobolVar sqlerrmc)
 {
      int    newCode=0;
      String newState="";
      String errorMsg="";
      // set the above 3 items as you wish (note that ex might be null)
      sqlcode.set (newCode);
      sqlstate.set(newState);
      sqlerrmc.set(errorMsg);
   }
}
```

To make isCOBOL use the above class for the setting of SQLCA fields, set

```
iscobol.esql.sqlca_handler=MySqlcaHandler
```

SQLDA

A SQLDA (SQL Descriptor Area) is a collection of variables that are required for execution of the SQL DESCRIBE statement, and can optionally be used by the PREPARE, OPEN, FETCH, EXECUTE, and CALL statements.

isCOBOL supports two flavors of SQLDA:

1. Oracle Pro*COBOL SQLDA implementation
2. IBM DB2 SQLDA implementation

Oracle Pro*COBOL SQLDA implementation

SQLDA is supported by default in compatibility with Oracle Pro*COBOL for the use on the Oracle database.

SQLDA has the following structure, that should be added to the Working-Storage Section of the program:

```
01 SELDSC.
   02 SQLDNUM PIC S9(9) COMP VALUE 100.
   02 SQLDFND PIC S9(9) COMP.
   02 SELDVAR OCCURS 100 TIMES.
      03 SELDV PIC S9(9) COMP.
      03 SELDFMT PIC S9(9 COMP.
      03 SELDVLN PIC S9(9) COMP.
      03 SELDFMTL PIC S9(4) COMP.
      03 SELDVTYP PIC S9(4) COMP.
      03 SELDI PIC S9(9) COMP.
      03 SELDH-VNAME PIC S9(9) COMP.
      03 SELDH-MAX-VNAMEL PIC S9(4) COMP.
      03 SELDH-CUR-VNAMEL PIC S9(4) COMP.
      03 SELDI-VNAME PIC S9(9) COMP.
      03 SELDI-MAX-VNAMEL PIC S9(4) COMP.
      03 SELDI-CUR-VNAMEL PIC S9(4) COMP.
      03 SELDFCLP PIC S9(9) COMP.
      03 SELDFCRCP PIC S9(9) COMP.
01 XSELDI.
   03 SEL-DI OCCURS 100 TIMES PIC S9(4) COMP.
01 XSELDIVNAME.
   03 SEL-DI-VNAME OCCURS 100 TIMES PIC X(80).
01 XSELDV.
   03 SEL-DV OCCURS 100 TIMES PIC X(80).
01 XSELDHVNAME.
   03 SEL-DH-VNAME OCCURS 100 TIMES PIC X(80).
01 BNDDSC.
   02 SQLDNUM PIC S9(9) COMP VALUE 100.
   02 SQLDFND PIC S9(9) COMP.
   02 BNDDVAR OCCURS 100 TIMES.
      03 BNDDV PIC S9(9) COMP.
      03 BNDDFMT PIC S9(9) COMP.
      03 BNDDVLN PIC S9(9) COMP.
      03 BNDDFMTL PIC S9(4) COMP.
      03 BNDDVTYP PIC S9(4) COMP.
      03 BNDDI PIC S9(9) COMP.
      03 BNDDH-VNAME PIC S9(9) COMP.
      03 BNDDH-MAX-VNAMEL PIC S9(4) COMP.
      03 BNDDH-CUR-VNAMEL PIC S9(4) COMP.
      03 BNDDI-VNAME PIC S9(9) COMP.
      03 BNDDI-MAX-VNAMEL PIC S9(4) COMP.
      03 BNDDI-CUR-VNAMEL PIC S9(4) COMP.
      03 BNDDFCLP PIC S9(9) COMP.
      03 BNDDFCRCP PIC S9(9) COMP.
01 XBNDDI.
   03 BND-DI OCCURS 100 TIMES PIC S9(4) COMP.
01 XBNDDIVNAME.
   03 BND-DI-VNAME OCCURS 100 TIMES PIC X(80).
01 XBNDDV.
   03 BND-DV OCCURS 100 TIMES PIC X(80).
01 XBNDDHVNAME.
   03 BND-DH-VNAME OCCURS 100 TIMES PIC X(80).
```

SQLDA fields have the following meaning.

SQLDNUM	Specifies the maximum number of select-list items or place-holders that can be included in DESCRIBE. Set this variable to the dimension of the descriptor tables before issuing a DESCRIBE command After the DESCRIBE, you must reset it to the actual number of variables in the DESCRIBE, which is stored in SQLDFND.
SQLDFND	After a DESCRIBE it holds the actual number of select-list items or place-holders found. If it is negative, the DESCRIBE command found too many select-list items or place-holders for the size of the descriptor.
SELDV | BNDDV	Contains the addresses of data buffers that store select-list or bind-variable values. Set the elements of SELDV or BNDDV using IWC$GET. Select descriptors (SELDV) store FETCHed select-list values. Bind descriptors (BNDDV) must be set before issuing the OPEN command.
SELDFMT | BNDDFMT	Contains the addresses of data buffers that store select-list or bind-variable conversion format strings.
SELDVLN | BNDDVLN	Contains the lengths of select-list variables or bind-variable values stored in the data buffers.
SELDFMTL | BNDDFMTL	Contains the lengths of select-list or bind-variable conversion format strings.
SELDVTYP | BNDDVTYP	Contains the datatype codes of select-list or bind-variable values. See Oracle External and Related COBOL Datatypes for detail.s
SELDI | BNDDI	Contains the addresses of data buffers that store indicator-variable values.
SELDH-VNAME | BNDDH-VNAME	Contains the addresses of data buffers that store select-list or place-holder names as they appear in dynamic SQL statements. Set the elements of SELDH- VNAME or BNDDH-VNAME using IWC$GET before issuing the DESCRIBE command.
SELDH-MAX-VNAMEL | BNDDH-MAX-VNAME	Contains the maximum lengths of the data buffers that store select-list or place-holder names. The buffers are addressed by the elements of SELDH-VNAME or BNDDH-VNAME. Set the elements of SELDH-MAX-VNAMEL or BNDDH-MAX-VNAMEL before issuing the DESCRIBE command.
SELDH-CUR-VNAMEL | BNDDH-CUR-VNAMEL	Contains the actual lengths of the names of the select-list or place-holder. The DESCRIBE statement sets the table of actual lengths to the number of characters in each select-list or place-holder name.
SELDI-VNAME | BNDDI-VNAME	Contains the addresses of data buffers that store indicator-variable names. You can associate indicator-variable values with select-list items and bind variables. However, you can associate indicator-variable names only with bind variables. You can use this table only with bind descriptors. Set the elements of BNDDI-VNAME using IWC$GET before issuing the DESCRIBE command.
SELDI-MAX-VNAMEL | BNDDI-MAX-VNAMEL	Contains the maximum lengths of the data buffers that store indicator-variable names. The buffers are addressed by the elements of SELDI-VNAME or BNDDI-VNAME. You can associate indicator-variable names only with bind variables. You can use this table only with bind descriptors. Set the elements BNDDI-MAX-VNAMEL(1) through BNDDI-MAX-VNAMEL(SQLDNUM) before issuing the DESCRIBE command. Each indicator-variable name buffer can have a different length.

SELDI-CUR-VNAMEL | BNDDI-CUR-VNAMEL	Contains the actual lengths of the names of the indicator variables. You can associate indicator-variable names only with bind variables. You can use this table only with bind descriptors.
SELDFCLP | BNDDFCLP	Reserved for future use.
SELDFCRCP | BNDDFCRCP	Reserved for future use.

Oracle External and Related COBOL Datatypes

Code	Oracle data type	COBOL data type
1	VARCHAR2	PIC X(n)
2	NUMBER	PIC X(n)
3	INTEGER	PIC S9(n) COMP
4	FLOAT	COMP-1 COMP-2
5	STRING	PIC X(n)
6	VARNUM	PIC X(n)
7	DECIMAL	PIC S9(n)V9(n) COMP-3
8	LONG	PIC X(n)
9	VARCHAR	PIC X(n) VARYING
11	ROWID	PIC X(n)
12	DATE	PIC X(n)
15	VARRAW	PIC X(n)
23	RAW	PIC X(n)
24	LONG RAW	PIC X(n)
68	UNSIGNED	Not supported
91	DISPLAY	PIC S9(n)V9(n) DISPLAY SIGN LEADING SEPARATE
94	LONG VARCHAR	PIC X(n)
95	LONG VARRAW	PIC X(n)
96	CHARF	PIC X(n)
97	CHARZ	PIC X(n)
98	CURSOR	SQL CURSOR

Handling NULL/Not NULL Datatypes

For every select-list column (not expression), DESCRIBE SELECT LIST returns a NULL/not NULL indication in the datatype table of the select descriptor. If the select-list column is constrained to be not NULL, the high-order bit of SELDVTYP datatype variable is clear; otherwise, it is set.

Before using the datatype in an OPEN or FETCH statement, if the NULL status bit is set, you must clear it.

You can use the SQLNUL function to find out if a column allows NULL datatypes and to clear the datatype's NULL status bit.

The -cp and -dz compiler options

Since pointers are stored in PIC S9(9) COMP data item, if you're using the -cp compiler option, you must use also the -dz compiler option.

IBM DB2 SQLDA implementation

SQLDA is supported in compatibility with IBM DB2 for the use on the IBM DB2 database.

In order to enable the compatibility with IBM DB2, use the -csdb2 compiler option.

SQLDA has the following structure, that should be added to the Working-Storage Section of the program:

```
      ******************************************************
      * SQL DESCRIPTOR AREA *
      ******************************************************
       01 SQLDA.
          02 SQLDAID PIC X(8) VALUE 'SQLDA '.
          02 SQLDABC PIC S9(8) COMPUTATIONAL VALUE 33016.
          02 SQLN PIC S9(4) COMPUTATIONAL VALUE 750.
          02 SQLD PIC S9(4) COMPUTATIONAL VALUE 0.
          02 SQLVAR OCCURS 1 TO 750 TIMES DEPENDING ON SQLN.
             03 SQLTYPE PIC S9(4) COMPUTATIONAL.
             03 SQLLEN PIC S9(4) COMPUTATIONAL.
             03 SQLDATA POINTER.
             03 SQLIND POINTER.
             03 SQLNAME.
                49 SQLNAMEL PIC S9(4) COMPUTATIONAL.
                49 SQLNAMEC PIC X(30).
```

SQLDA fields are set by the database manager after a DESCRIBE or a PREPARE statement and they have the following meaning.

SQLDAID	Contains the string 'SQLDA '
SQLDABC	Length of the SQLDA
SQLN	Unchanged in this context
SQLD	Number of columns described by occurrences of SQLVAR
SQLTYPE	The data type of the column. See SQLTYPE values below for the list of possible values
SQLLEN	The length attribute of the column. If the data is decimal, the first byte contains the precision; the second byte contains the scale.
SQLDATA	Not used
SQLIND	Not used
SQLNAME	The unqualified name of the column. This is a varchar field: SQLNAMEL contains the length of the value, while SQLNAMEC contains the value.

SQLDA fields can also set by the user prior to executing a FETCH statement, an OPEN statement, a CALL statement or an EXECUTE statement. In these cases they have the following meaning.

SQLDAID	Contains the string 'SQLDA '
SQLDABC	Length of the SQLDA
SQLN	Total number of occurrences of SQLVAR provided in the SQLDA
SQLD	Number of occurrences of SQLVAR entries in the SQLDA that are used when executing the statement. It must be set to a value greater than or equal to zero and less than or equal to SQLN
SQLTYPE	The data type of the host variable and whether an indicator variable is provided. See SQLTYPE values below for the list of possible values
SQLLEN	The length attribute of the host variable
SQLDATA	Contains the address of the host variable
SQLIND	Contains the address of the indicator variable. It's ignored if there is no indicator variable (as indicated by an even value of SQLTYPE)
SQLNAME	Not used

Note - due to the presence of POINTER data items that can be reused and included in arithmetic expressions, programs including SQLDA should be compiled with the -cp option and, if you're on a 64-bit platform, with -d64 option as well. If -cp is not already used for other purposes and you wish to avoid using it only due to SQLDA, contact Veryant to have some advice on alternate approaches that don't require arithmetic operations on POINTER data items.

SQLTYPE values

Value without indicator	Value with indicator	Type
384	385	Date
388	389	Time
392	393	Timestamp
396	397	DataLink
404	405	BLOB
408	409	CLOB
412	413	DBCLOB
448	449	Varying-length character string
452	453	Fixed-length character string
456	457	Long varying-length character string
480	481	Floating point
484	485	Packed decimal
488	489	Zoned decimal
492	493	Big integer
496	497	Large integer
500	501	Small integer
904	905	ROWID
908	909	Varying-length binary string
912	913	Fixed-length binary string
988	989	XML

SQLJ

Introduction

SQLJ enables programmers to embed SQL statements in Java code in a way that is compatible with the Java design philosophy. A SQLJ program is a Java program containing embedded SQL statements that comply with the International Standardization Organization (ISO) standard SQLJ Language Reference syntax. The standard covers only static SQL operations, which are predefined SQL operations that do not change in real time while a user runs the application. Typical applications contain more static SQL operations than dynamic SQL operations.

SQLJ consists of a translator and a run-time component and is smoothly integrated into your development environment. You can run the translator to translate and compile the code in a single step using the *sqlj* utility. The translation process replaces embedded SQL statements with calls to the SQLJ run time, which processes the SQL statements. In ISO standard SQLJ this is typically, but not necessarily, performed through calls to a JDBC driver. To access a database, you would typically use a JDBC driver. When you run the SQLJ application, the run time is started to handle the SQL operations.

The SQLJ translator is conceptually similar to other precompilers and enables you to check SQL syntax, verify SQL operations against what is available in the schema, and check the compatibility of Java types with corresponding database types. In this way, you can catch errors during development rather than a user catching the errors at run time.

SQLJ is currently supported by Oracle and IBM DB2.

How to generate SQLJ code through the compiler

To generate SQLJ code for your embedded SQL, use the -sqlj compiler option.

The compiler produces an intermediate .sqlj source file and automatically invokes the SQLJ translsator to compile the .sqlj source file to class. In the end you obtain a class and a profile file produced by the SQLJ translator. The .sqlj source file is removed from disc, unless you have the -jj option in your compiler command line.

The SQLJ translator (*sqlj.exe* on Windows, *sqlj* on Unix/Linux) must be available in the Path, otherwise the above process fails and you obtain only the .sqlj source file, that you will have to compile later by running the SQLJ translator yourself.

If the SQLJ translator is not available in the Path, you can inform the compiler about the *sqlj* utility location through the following configuration setting:

```
iscobol.compiler.sqlj=/path/to/sqlj_executable
```

The iscobol.compiler.sqlj.options * configuration setting allows you to specify some *sqlj* command options. Refer to the SQLJ documentation of your database for the list of available options.

SQLJ usage example for IBM DB2

The following steps are necessary to compile and run the isCOBOL's ESQL sample on IBM DB2 using SQLJ:

1. compile the program with the -sqlj option

```
iscc -sqlj ESQL-SAMPLE.cbl
```

2. run *db2sqljcustomize* to to create a customized serialized profile

```
db2sqljcustomize -user richler -password mordecai
     -url jdbc:db2:/server:50000/sample -collection duddy
     -bindoptions "EXPLAIN YES" ESQL_SAMPLE_SJProfile0.ser
```

3. run *db2sqljbind* to bind DB2 packages for the serialized profile that was previously customized

```
db2sqljbind -user richler -password mordecai
     -url jdbc:db2://server:50000/sample -bindoptions "EXPLAIN YES"
     ESQL_SAMPLE_SJProfile0.ser
```

4. run the program

```
iscrun ESQL_SAMPLE
```

Note - *db2sqljcustomize* and *db2sqljbind* were not available on old DB2 versions. The deprecated *db2profc* utility can be used instead on old DB2 versions.

Working on multiple connection simultaneously

The Embedded SQL syntax allows you to open more than one connection to the same database or to different databases and perform every SQL query on a specific connection. In this kind of scenario every connection must be given a unique name.

There are two ways to create a connection with a name:

- Using a Format 1 CONNECT statement with the AS clause, e.g.

```
connect-ora.
    set environment "jdbc.driver"
        to "oracle.jdbc.OracleDriver"
    set environment "jdbc.url"
        to "jdbc:oracle:thin:@192.168.1.6:1521:"
    exec sql
        connect to :orcl-db as orcl
                user :orcl-user using :orcl-pwd

    end-exec
    .
connect-mssql.
    set environment "jdbc.driver"
        to "com.microsoft.sqlserver.jdbc.SQLServerDriver"
    set environment "jdbc.url"
        to "jdbc:sqlserver://192.168.1.7:1433;encrypt=false;DatabaseName="
    exec sql
        connect to :msql-db as msql
                user :msql-user using :msql-pwd
    end-exec
    .
```

- Using a Format 2 CONNECT statement, preceded by a Format 4 DECLARE statement, e.g.

```
connect-ora.
    set environment "jdbc.driver"
        to "oracle.jdbc.OracleDriver"
    set environment "jdbc.url"
        to "jdbc:oracle:thin:@192.168.1.6:1521:"
    exec sql
       declare orcl database
    end-exec
    exec sql
        connect :orcl-user identified by :orcl-pwd
                at orcl using :orcl-db
    end-exec
    .
connect-mssql.
    set environment "jdbc.driver"
        to "com.microsoft.sqlserver.jdbc.SQLServerDriver"
    set environment "jdbc.url"
        to "jdbc:sqlserver://192.168.1.7:1433;encrypt=false;DatabaseName="
    exec sql
       declare msql database
    end-exec
    exec sql
        connect :msql-user identified by :msql-pwd
               at msql using :msql-db
    end-exec
```

When more than one connection is available and every connection has an unique name, the program can perform queries on one of these named connections.

There are two ways to perform a query on a given connection:

Switching the active connection with a SET CONNECTION statement before the query, e.g.

```
exec sql
   set connection msql
end-exec
exec sql
   insert into cities (zip_code, city_name)
          values (:city-zip, :city-name)
end-exec
```

Specifying the connection name on the query through the AT clause, e.g.

```
exec sql at msql
   insert into cities (zip_code, city_name)
          values (:city-zip, :city-name)
end-exec
```

The below programs show how to transfer data from a table in an Oracle database to a table with the same fields in a Microsoft SQL Server database. The two programs use different syntaxes but produce the same result.

Example 1 (using SET CONNECTION)

```
       program-id. oratomssql.

       working-storage section.
           exec sql include sqlca end-exec.
           exec sql begin declare section end-exec.
       77 orcl-user  pic x(6) value "system".
       77 orcl-pwd   pic x(5) value "admin".
       77 orcl-db    pic x(2) value "xe".
       77 msql-user pic x(2) value "sa".
       77 msql-pwd  pic x(7) value "manager".
       77 msql-db   pic x(6) value "master".
       77 city-zip  pic 9(5).
       77 city-name pic x(32).
           exec sql end declare section end-exec.

       procedure division.
       main.
           exec sql
              whenever sqlerror go to abend
           end-exec
           .
       connect-ora.
           set environment "jdbc.driver"
               to "oracle.jdbc.OracleDriver"
           set environment "jdbc.url"
               to "jdbc:oracle:thin:@192.168.1.6:1521:"
           exec sql
               connect to :orcl-db as orcl
                       user :orcl-user using :orcl-pwd

           end-exec
           .
       connect-mssql.
           set environment "jdbc.driver"
               to "com.microsoft.sqlserver.jdbc.SQLServerDriver"
           set environment "jdbc.url"
               to "jdbc:sqlserver://192.168.1.7:1433;encrypt=false;DatabaseName="
           exec sql
               connect to :msql-db as msql
                       user :msql-user using :msql-pwd
           end-exec
           .
       transfer-data.
           exec sql
              set connection orcl
           end-exec
           exec sql
              declare ora-data cursor for
                    select zip_code, city_name from cities
           end-exec
           exec sql
              open ora-data into :city-zip, :city-name
           end-exec
           perform until exit
              exec sql
                 fetch next ora-data
              end-exec
```

```
            if sqlcode = 100
               exit perform
            end-if
            exec sql
               set connection msql
            end-exec
            exec sql
               insert into cities (zip_code, city_name)
                      values (:city-zip, :city-name)
            end-exec
            exec sql
               set connection orcl
            end-exec
        end-perform.
        exec sql
           close ora-data
        end-exec
        .
    disconnect-exit.
        exec sql
           disconnect all
        end-exec
        exit program
        stop run
        .
    abend.
        display sqlcode
        display sqlerrmc
        exit program
        stop run
        .
```

Example 2 (using AT DATABASE)

```
       program-id. oratomssql.

       working-storage section.
           exec sql include sqlca end-exec.
           exec sql begin declare section end-exec.
       77 orcl-user  pic x(6) value "system".
       77 orcl-pwd   pic x(5) value "admin".
       77 orcl-db    pic x(2) value "xe".
       77 msql-user pic x(2) value "sa".
       77 msql-pwd  pic x(7) value "manager".
       77 msql-db   pic x(6) value "master".
       77 city-zip  pic 9(5).
       77 city-name pic x(32).
           exec sql end declare section end-exec.

       procedure division.
       main.
           exec sql
              whenever sqlerror go to abend
           end-exec
           .
       connect-ora.
           set environment "jdbc.driver"
               to "oracle.jdbc.OracleDriver"
           set environment "jdbc.url"
               to "jdbc:oracle:thin:@192.168.1.6:1521:"
           exec sql
              declare orcl database
           end-exec
           exec sql
               connect :orcl-user identified by :orcl-pwd
                       at orcl using :orcl-db
           end-exec
           .
       connect-mssql.
           set environment "jdbc.driver"
               to "com.microsoft.sqlserver.jdbc.SQLServerDriver"
           set environment "jdbc.url"
               to "jdbc:sqlserver://192.168.1.7:1433;encrypt=false;DatabaseName="
           exec sql
              declare msql database
           end-exec
           exec sql
               connect :msql-user identified by :msql-pwd
                      at msql using :msql-db
           end-exec
           .
       transfer-data.
           exec sql at orcl
              declare ora-data cursor for
                    select zip_code, city_name from cities
           end-exec
           exec sql at orcl
              open ora-data into :city-zip, :city-name
           end-exec
           perform until exit
```

```
            exec sql at orcl
               fetch next ora-data
            end-exec
            if sqlcode = 100
               exit perform
            end-if
            exec sql at msql
               insert into cities (zip_code, city_name)
                      values (:city-zip, :city-name)
            end-exec
        end-perform.
        exec sql at orcl
           close ora-data
        end-exec
        .
    disconnect-exit.
        exec sql
           disconnect all
        end-exec
        exit program
        stop run
        .
    abend.
        display sqlcode
        display sqlerrmc
        exit program
        stop run
        .
```

Chapter 10

Embedded SQL Statements

NOTE - EXEC SQL and END-EXEC are shown in the statement format to improve clarity. EXEC SQL and END-EXEC are not part of the statements.

ALLOCATE

The ALLOCATE statement allows you to define and allocate host variables.

General Format

```
EXEC SQL [ AT Database ]

  ALLOCATE Host-Variable [ Variable-Type ]

END-EXEC
```

Syntax Rules

1. *Host-Variable* must be defined as USAGE HANDLE in DATA DIVISION.
2. *Variable-Type* is a numeric data-item or literal that specifies the type of the item to allocate. It must reflect one of the java.sql.Types constant values (http://java.sun.com/javase/6/docs/api/constant-values.html#Types.sql). If omitted, iscobol.jdbc.allocate_type setting is considered.

General Rules

1. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.

Examples

Allocate is an Oracle specific SQL stmt. This sample allocates a cursor handle, uses it and releases it at the end.

```
exec sql allocate :cur-hndl end-exec.

exec sql execute
   begin
        open :cur-hndl for select * from emp;
   end;
end-exec.

exec sql fetch :cur-hndl into :empno,:ename
end-exec.

exec sql free :cur-hndl
end-exec.
```

ALTER

The ALTER statement allows you to modify a resource.

General format

```
EXEC SQL [ AT Database ]

  ALTER Options

END-EXEC
```

Syntax rules

1. *Options* is passed to the driver without further checks. Refer to the database documentation for detailed syntax. Syntax errors, if any, are returned at runtime.

General Rules

1. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.

Examples

Alter a table to add its primary key

```
exec sql
   alter table customers add primary key (cust_code)
end-exec
```

BEGIN

The BEGIN ... END statement gives the statements defined within the BEGIN and END keywords the status of a single statement.

General Format

```
EXEC SQL

  BEGIN Options

END-EXEC
```

Syntax rules

The BEGIN statement is a processed as a comment statement. It can be used only in DATA DIVISION.

Examples

Show the begin of the declare section

```
exec sql begin declare section end-exec
```

CALL

The CALL statement allows you to invoke stored procedures.

Format 1

```
EXEC SQL [ AT Database ]

  CALL { Procedure-Name  } ( [ Parameters ] ) [ INTO Host-Variable-2 ]
       { Host-Variable-1 }

END-EXEC
```

Format 2

```
EXEC SQL [ AT Database ]

  [ Host-Variable-2 = ] CALL { Procedure-Name  } ( [ Parameters ] )
                             { Host-Variable-1 }

END-EXEC
```

Syntax Rules

1. The stored procedure name can be specified directly in the statement or through an host variable.
2. Multiple parameters must be separated by commas.
3. Each parameter name might be followed by the parameter type. The type can be IN, OUT or INOUT.

General Rules

1. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.
2. Only one destination Host-Variable is allowed. If the stored procedure returns multiple fields or multiple rows, rely on a cursor defined using a Format 1 DECLARE statement.

3. If the parameter type is not specified, the Compiler sets the type according to the HOSTVAR Directive directive or by the iiscobol.compiler.esql.procedure.ProcedureName configuration property. If neither the HOSTVAR Directive nor the iscobol.compiler.esql.procedure.ProcedureName configuration property is present, then the type is set at runtime according to the iscobol.esql.default_param_type configuration property, that is INOUT by default.

Examples

Call a Stored Procedure that updates values on a totals table (the logic of the update is DB Server based and stored)

```
exec sql
  call update_totals()
end-exec
```

Call a Stored Procedure with an input parameter and an output parameter, storing the result in a destination data item

```
exec sql
  call proc1(:p1 IN, :p2 OUT) into :exit-status
end-exec
```

CLOSE

The CLOSE statement closes an open cursor. Once closed, the cursor cannot be used for further processing.

The COMMIT, ROLLBACK, and DISCONNECT statements close all open cursors.

General format

```
EXEC SQL [ AT Database ]

  CLOSE Cursor-Name

END-EXEC
```

Syntax rules

1. *Cursor-Name* is a Nonnumeric Literal, as defined in the Definitions section in the Preface of this document.

General rules

1. *Cursor-Name* must be previously defined by a DECLARE statement.
2. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.

Examples

Close cursor

```
exec sql close cust_cur end-exec
```

COMMIT

The COMMIT statement commits the current transaction and automatically starts a new transaction.

General format

```
EXEC SQL [ AT Database ]

  COMMIT { WORK        } [ RELEASE ]
         { TRAN        }
         { TRANSACTION }
END-EXEC
```

General Rules

1. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.
2. WORK, TRAN and TRANSACTION are synonymous and are assumed if omitted.
3. The RELEASE option frees all resources (locks and cursors) held by the program and logs off the database.
4. The iscobol.jdbc.autocommit (boolean) configuration setting must be set to false, otherwise all statements are automatically commited. With iscobol.jdbc.autocommit (boolean) set to false, a transaction is started as soon as you connect to the database. The COMMIT statement allows you to confirm that transaction and immediately start a new one.

Examples

Commit transaction

```
exec sql commit work end-exec
```

CONNECT

The CONNECT statement allows you to connect to a database.

Format 1

```
EXEC SQL

  CONNECT [ TO [ { Database-Name } ] [ AS Connection-Name ] ]
                 { DEFAULT       }

          [ USER User-Name [ USING Password ] ]

END-EXEC
```

Format 2

```
EXEC SQL

  CONNECT  [ {  User-Name IDENTIFIED BY Password }[ AT Database ] USING Database-
Name ]
            { Usr-Pwd                           }
END-EXEC
```

Format 3

```
EXEC SQL

  CONNECT  USING Connection-String

END-EXEC
```

Syntax rules

1. *Database-Name*, *User-Name*, *Usr-Pwd, Password* , *DBString*, *Connection-Name* and *Connection-String* are Nonnumeric Literals, as defined in the Definitions section of the Preface of this document or Host Variables.

General rules

1. The connect statement must precede all statements that access the database.
2. In Format 1, when *Database-Name* is omitted or DEFAULT is specified, no database name is passed to the jdbc driver. If USER and USING are omitted too, then no parameter is passed to the jdbc driver, assuming that all the necessary information is provided by the iscobol.jdbc.url configuration property.
3. In Format 2, when the USING phrase is omitted no database name is passed to the jdbc driver. If user and password credentials are omitted too, then no parameter is passed to the jdbc driver, assuming that all the necessary information is provided by the iscobol.jdbc.url configuration property.
4. *Connection-String* should contain all the necessary information for the connection to the database, e.g. driver class and connection URL. The following formats are supported:

   - DRIVER=<driver-class>;URL=<connection-url>
   - DRIVER=<driver-class>
   - URL=<connection-url>
   - <connection-url>

   If either the driver class or the connection URL is not provided, the runtime retrieves the information from the iscobol.jdbc.driver and the iscobol.jdbc.url configuration properties.

5. When the AS phrase is omitted, the connection name is the same as the database name.
6. Once connected you can retrieve the connection handle by invoking the getCurrConnection() method of the com.iscobol.rts.EsqlRuntime object. This method returns a java.sql.Connection that can be passed to other java programs. See EsqlRuntime (com.iscobol.rts.EsqlRuntime) for more information.
7. *Usr-Pwd* contains both UserName and Password separated by "/", for example "scott/tiger".
8. *Database* must be previously defined using a Format 4 DECLARE statement.
9. If the iscobol.jdbc.datasource configuration property specifies a valid class, then the connect() method of that class is invoked to gain connection. If the property is not set, then the runtime invokes the Java SQL DriverManager to gain connection. The following snippet shows a prototype of a valid class

```
import java.sql.*;
import javax.sql.*;
public class myDataSource implements com.iscobol.rts.MyDataSource {
   public Connection connect (String dsn, String usr, String pwd)
                                                    throws SQLException {
      Connection conn = getMyConnection(usr, pwd);
      return conn;
   }
   private Connection getMyConnection(String usr, String pwd) {
      // do the necessary operation to get the connection
   }
}
```

Note - The parameters *dsn*, *user* and *passwd* are the ones used in the CONNECT statement. If the program performs just CONNECT because database name and login credentials were put in the JDBC URL, then the class does not receive any parameter.

To make isCOBOL use the above class for connection, set

```
iscobol.jdbc.datasource=myDataSource
```

10. *Connection-Name* can be referenced by the SET CONNECTION statement.

Examples

Format 1 - Connect to the ctree database

```
move "ctreeSQL" to dbname
move "admin"    to userid
move "ADMIN"    to passwd
exec sql
   connect to :dbname
         user :userid
        using :passwd
end-exec
```

CONNECT RESET

The CONNECT RESET statement closes the connection with the database.

General format

```
EXEC SQL [ AT Database ]

  CONNECT RESET [ { DEFAULT } ]
                  { CURRENT }
                  { ALL     }

END-EXEC
```

Syntax rules

1. CURRENT and DEFAULT are synonymous.

General rules

1. Unless the ALL prhase is specified, the current databae connection is closed.
2. When the ALL phrase is specified, all the active connections are closed.
3. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.

Examples

Disconnect from the current connection

```
exec sql connect reset end-exec
```

CREATE

The CREATE statement allows you to create tables and indices.

General format

```
EXEC SQL [ AT Database ]

  CREATE Options

END-EXEC
```

Syntax rules

1. *Options* is passed to the driver without further checks. Refer to the database documentation for detailed syntax. Syntax errors, if any, are returned at runtime.

General Rules

1. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.

Examples

Create a table

```
exec sql
   create table customers
          (cust-code numeric(4) not null,
           cust-name char(30))
end-exec
```

DEALLOCATE

The DEALLOCATE statement removes a cursor reference.

General format

```
EXEC SQL [ AT Database ]

  DEALLOCATE Options

END-EXEC
```

Syntax rules

1. The DEALLOCATE statement is processed as a comment statement.

Examples

Show the end of the declare section

```
exec sql deallocate cust_cur end-exec
```

DECLARE

The DECLARE statement associates a cursor name with a SELECT statement or a CALL statement. It also allows you to define tables and database entities.

Format 1

```
EXEC SQL [ AT Database ]

  DECLARE Cursor-Name [ { SENSITIVE   } SCROLL ] CURSOR
                        { INSENSITIVE }
      [ WITH [NO] HOLD ]
      [ { WITHOUT } RETURN ] ]
        { WITH    }
      [ { WITHOUT } ROWSET POSITIONING } ]
        { WITH    }
                                        FOR { Prepared-Statement }
                                            { Select-Statement   }
                                            { Call-Statement     }

END-EXEC
```

Format 2

```
EXEC SQL [ AT Database ]

  DECLARE Prepared-Statement STATEMENT

END-EXEC
```

Format 3

```
EXEC SQL [ AT Database ]

  DECLARE Table-Name TABLE

END-EXEC
```

Format 4

```
EXEC SQL

  DECLARE Database DATABASE

END-EXEC
```

Format 5

```
EXEC SQL [ AT Database ]

  DECLARE GLOBAL TEMPORARY TABLE Table-Name

END-EXEC
```

Syntax rules

1. *Cursor-Name*, *Database, Procedure-Name* and *Prepared-Statement* are Nonnumeric Literals, as defined in the Definitions section of the Preface of this document.

2. *Select-Statement* is a complete SELECT statement.
3. *Call-Statement* is a CALL statement without the INTO clause.

General rules

1. The DECLARE statement must appear in the source before any other statement referencing *Cursor-Name, Prepared-Statement, Table-Name* and *Database*.
2. The DECLARE statement can appear in either the working-storage section or in the procedure division.
3. The DECLARE statement doesn't set SQLCA entries.
4. *Prepared-Statement* must be previously defined by a PREPARE statement.
5. *Call-Statement* must not use the INTO clause. The INTO clause must be used on OPEN or FETCH of the cursor.
6. The behavior of the WITH clause is database dependent.
    a. When the NO phrase is not specified, the cursor may be closed as a consequence of a commit operation.
    b. When the NO phrase is specified, the cursor is closed as a consequence of a commit operation.
7. Format 3 is supported for compatibility and is treated as commentary.
8. Format 4 defines a named connection. It should be used before a Format 2 CONNECT statement.
9. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.
10. A Format 5 DECLARE statement is treated as comment if it's found in the Working-Storage Section. If it appears in the Procedure Division, it's passed as is to the database.
11. *Cursor-Name* and *Prepared-Statement* are bound to the program object instance.
    A cursor or a statement declared in a standard program is available only inside the program.
    In object oriented programming, a cursor or a statement declared in an Object is shared between all the methods of the Object. The same doesn't apply to Factory; cursors and statements cannot be shared between Factory methods, they must be declared and used in the same method.
12. The SCROLL clause overrides the iscobol.jdbc.cursor.type * configuration setting.
    SENSITIVE SCROLL allows the cursor to move forward and backwards through the data. Changes made while the cursor is open are immediately available. It provides a dynamic view of the underlying data to which the cursor refers.
    INSENSITIVE SCROLL allows the cursor to move forward and backward through the data. Changes made while the cursor is open are ignored. It provides a static view of the underlying data to which the cursor refers.
13. WITHOUT ROWSET POSITIONING specifies that the cursor can be used only with row-positioned FETCH statements. The cursor is to return a single row for each FETCH statement and the FOR n ROWS clause cannot be specified on a FETCH statement for this cursor.
    WITH ROWSET POSITIONING specifies that the cursor can be used with either row-positioned or rowset-positioned FETCH statements. This cursor can be used to return either a single row or multiple rows, as a rowset, with a single FETCH statement.
14. WITHOUT RETURN doesn't expose the java.sql.ResultSet of the cursor to the caller program.
    WITH RETURN exposes the java.sql.ResultSet of the cursor to the caller program. This feature is particularly useful to share the ResultSet with a Java caller program. The caller program must invoke the *registerResultSets* method of the com.iscobol.rts.EsqlRuntime before calling the COBOL program that uses cursors with the WITH RETURN clause. The COBOL program must just open the cursor without performing any fetch and without closing it before exiting. The following Java code snippets demonstrates the good practice:

Java caller

```
import com.iscobol.java.IsCobol;
import com.iscobol.rts.EsqlRuntime;
import java.sql.ResultSet;
...
        //enable resultset collection
        IsCobol.registerResultSets();
        //call the COBOL program that uses cursors WITH RETURN
        IsCobol.call ("CBLPROG", new Object[0]);
        //
retrieve the array of ResultSet object that correspond to the cursors WITH RETURN that
 are still open
        ResultSet[] rs = IsCobol.getResultSets();
...
```

COBOL callee

```
       PROGRAM-ID. CBLPROG.
       ...
           EXEC SQL DECLARE CUR_RET_A CURSOR WITH RETURN FOR
                SELECT C1, C2
                  INTO :WK-C1, :WK-C2
                  FROM TBL
                  WHERE C2 = 'A'
           END-EXEC.
       ...
           EXEC SQL OPEN CUR_RET_A END-EXEC.
           GOBACK.
```

Examples

Format 1 - Declare, open and fetch a cursor

```
exec sql
   declare cust_cur  cursor for select * from customers
end-exec

exec sql
   open cust_cur
end-exec

perform until 1 = 2
   exec
      sql fetch next cust_cur into :ws-cust-code, :ws-cust-name
   end-exec

   display "code: " ws-cust-code " name: " ws-cust-name
   if sqlcode = 100
      exit perform
   end-if

end-perform

exec sql
   close cust_cur
end-exec
```

Format 1 - Declare a cursor to intercept the result of a stored procedure that returns a resultset. Refer to the snippet above for information on how to read the content of the cursor

```
exec sql
   declare cities cursor for call locateStores(:userState)
end-exec
```

DELETE

The DELETE statement removes rows from a database table.

General format

```
EXEC SQL [ AT Database ]

  [ FOR Iterations ]

  DELETE Options [ WHERE { Search-Condition                                    } ]
                         { CURRENT OF Cursor-Name [ FOR ROW Row-Num OF ROWSET ] }

                 [ RETURNING Field ... INTO HostVariable ... ]

END-EXEC
```

Syntax rules

1. *Iterations* can be either a host variable or a numeric literal. It specifies the number of rows to be processed.

2. *Options* is passed to the driver without further checks. Refer to the database documentation for detailed syntax. Syntax errors, if any, are returned at runtime.
3. *Search-Condition* is composed of predicates of various kinds, optionally combined using parentheses and logical operators. Its syntax is database dependent, therefore only Host Variables are handled and no further syntax checking is performed. Syntax errors, if any, are returned at runtime.
4. Cursor-Name must be previously defined by a DECLARE statement.
5. *Field* is a alphanumeric literal.
6. *Row-Num* is a numeric literal or host variable.
7. *Host-Variable* is a host variable.

General rules

1. When *Search-Condition* is specified, only rows matching it are deleted.
2. The FOR clause limits the number of times the statement is executed when *Search-Condition* contains array host variables If you omit this clause,it executes the statement once for each component of the smallest array.
3. When *Search-Condition* is not specified, all rows are deleted.
4. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.
5. CURRENT OF allows you to delete the last fetched record in the active cursor.
   When this syntax is used
   - the statement associated to the cursor should not use "SELECT *" and
   - the statement associated to the cursor should query from a single table.

   Otherwise a "cursor is not updatable" error may occur.
6. FOR ROW n OF ROWSET specifies which row of the current rowset is to be deleted. The corresponding row of the rowset is deleted, and the cursor remains positioned on the current rowset. If the rowset consists of a single row, or all other rows in the rowset have already been deleted, then the cursor is positioned before the next rowset of the result table. If there is no next rowset, the cursor is positioned after the last rowset.
7. The RETURNING clause allows you to receive the updated fields value into host variables. This clause is currently supported only on the Oracle database and requires the -csora compiler option.

Examples

Delete 2 records by specific key

```
exec sql
   delete from customers where  cust-code = 2 or cust-code = 4
end-exec
```

Declare, open and fetch a cursor. Delete records where the field cust-name is empty.

```
exec sql
   declare cust_cur  cursor for select cust_code, cust_name from customers
end-exec

exec sql
   open cust_cur
end-exec

perform until 1 = 2
   exec
      sql fetch next cust_cur into :ws-cust-code, :ws-cust-name
   end-exec

   if ws-cust-name = space
      exec sql
           delete from customers where current of cust_cur
      end-exec
   end-if

   if sqlcode = 100
      exit perform
   end-if

end-perform

exec sql
   close cust_cur
end-exec
```

DESCRIBE

The DESCRIBE statement allows you to obtain information about parameters and other features of a prepared object before you execute it.

Format 1

```
EXEC SQL [ AT Database ]

  DESCRIBE { BIND VARIABLES  } FOR Prepared-Statement INTO { BNDDSC }
           { SELECT LIST     }                            { SELDSC }

END-EXEC
```

Format2

```
EXEC SQL [ AT Database ]

  DESCRIBE { INPUT  } Prepared-Statement INTO Sql-Descriptor
           { OUTPUT }

END-EXEC
```

Syntax rules

Format 1

1. BNDDSC and SELDSC are group items defined in the Oracle Pro*COBOL SQLDA implementation.
2. *Prepared-Statement* is a Nonnumeric Literal, as defined in the Definitions section of the Preface of this document.

Format 2

1. The -csdb2 Compiler option must be used in order to compile this statement.
2. *Prepared-Statement* is a Nonnumeric Literal, as defined in the Definitions section of the Preface of this document.
3. *Sql-Descriptor* is the group item that identifies the IBM DB2 SQLDA implementation.

General rules

Format 1

1. DESCRIBE BIND VARIABLES puts descriptions of bind variables into a bind descriptor. It must follow the PREPARE statement but precede the OPEN.
2. DESCRIBE SELECT LIST puts descriptions of select-list items into a select descriptor. It must follow the OPEN statement but precede the FETCH statement.

Format 2

1. The input INPUT keyword lets you receive information about input parameters.
2. The input OUTPUT keyword lets you receive information about output parameters.

Examples

Prepare a command and get information about it in compatibility with IBM DB2

```
exec sql
   prepare cmd from "select c1 from tbl where c1 = ?"
end-exec.
exec sql describe input cmd into :sqlda end-exec.
```

DISCONNECT

The DISCONNECT statement closes the connection with the database.

General format

```
EXEC SQL [ AT Database ]

  DISCONNECT [ { Connection-Name } ]
             { CURRENT          }
             { ALL              }

END-EXEC
```

Syntax rules

1. *Connection-Name* is a Nonnumeric Literal, as defined in the Definitions section of the Preface of this document.
2. *Connection-Name* can be referenced by the SET CONNECTION statement

General rules

1. CURRENT is default.
2. When the ALL phrase is specified, the application is disconnected from all connected databases.
3. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.

Examples

Disconnect from the current connection

```
exec sql disconnect end-exec
```

DROP

The DROP statement allows you to remove resources.

General format

```
EXEC SQL [ AT Database ]

  DROP Options

END-EXEC
```

Syntax rules

1. *Options* is passed to the driver without further checks. Refer to the database documentation for detailed syntax. Syntax errors, if any, are returned at runtime.

General Rules

1. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.

Examples

Remove the customers table

```
exec sql
   drop table customers
end-exec
```

END

The BEGIN ... END statement gives the statements defined within the BEGIN and END keywords the status of a

single statement.

General Format

```
EXEC SQL

  END Options

END-EXEC
```

Syntax rules

1. The END statement is a processed as a comment statement. It can be used only in DATA DIVISION.

Examples

Show the end of the declare section

```
exec sql end declare section end-exec
```

EXECUTE

The EXECUTE statement executes a SQL statement.

Format 1

```
EXEC SQL [ AT Database ]

  [ FOR Iterations ]

  EXECUTE Prepared-Statement [ USING Host-Variable, ... ] [ INTO Host-Variable, ... ]

END-EXEC
```

Format 2

```
EXEC SQL [ AT Database ]

  [ FOR Iterations ]

  EXECUTE IMMEDIATE Statement [ USING Host-Variable, ... ] [ INTO Host-Variable, ... ]

END-EXEC
```

Format 3

```
EXEC SQL [ AT Database ]

  EXECUTE  {BEGIN   }   [Plsql]
           {DECLARE }

END-EXEC
```

Format 4

```
EXEC SQL [ AT Database ]

  EXECUTE Prepared-Statement [ { USING DESCRIPTOR } Sql-Descriptor ]
                             { INTO DESCRIPTOR  }
END-EXEC
```

Syntax rules

1. *Iterations* can be either a host variable or a numeric literal. It specifies the number of rows to be processed.
2. *Prepared-Statement* is a Nonnumeric Literal, as defined in the Definitions section of the Preface of this document.
3. *Host-Variable* is a host variable.
4. *Statement* can be a host variable or a Nonnumeric Literal containing a complete SQL Statement.
5. *Plsql* is a Nonnumeric Literal containing a PL/SQL code.
6. *Sql-Descriptor* is a SQLDA structure. This syntax is compiled only under the -csdb2 option.

General rules

1. *Prepared-Statement* must be previously defined by a PREPARE statement.

2. The FOR clause limits the number of times the statement is executed when the USING clause contains array host variables If you omit this clause,it executes the statement once for each component of the smallest array.
3. The isCOBOL compiler does not analyze PL/SQL language. It only manages the host variables. The host variables management is conditioned by the HOSTVAR Directive, the iscobol.compiler.esql.procedure.ProcedureName configuration property and the iscobol.esql.default_param_type configuration property.
4. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.
5. If the iscobol.esql.prepare_handler * configuration property specifies a valid class, then the queryDecoder() method of that class is invoked to confirm or alter the statement. The runtime will then execute the new statement returned by that method. The following snippet shows a prototype of a valid class

```
import com.iscobol.types.CobolVar;
import com.iscobol.rts.EsqlPrepareHandler;

public class MyQueryHandler implements EsqlPrepareHandler {

   public void queryDecoder(CobolVar query) {
      String newQuery = query.toString();
      // alter the newQuery string as you wish
       query.set(newQuery);
   }
}
```

To make isCOBOL use the above class for every EXECUTE, set

```
iscobol.esql.prepare_handler=MyQueryHandler
```

Examples

Format 1 - Execute a prepared statement to count records that meet a criteria

```
*> count_recs does not need to be defined prior to
*> their use in the prepare statement
*> min-key and the-count could be pic 9(4) each

exec sql
     prepare count_recs from
        "select count(*) from cust_table where cust_code > ?"
end-exec
move 1990 to min-key
exec sql
     execute count_recs using :min-key
             into :the-count
end-exec

display "Count of records with key > " min-key " : " the-count
```

Format 2 - Execute inmmediate a statement to count records that meet a criteria

```
move 1990 to min-key
exec sql
     execute immediate
             "select count(*) from cust_table where cust_code > ?"
             using :min-key
             into :the-count
end-exec

display "Count of records with key > " min-key " : " the-count
```

Format 3 - To be used with Oracle Databases only using PL/SQL

```
exec sql
  execute
     DECLARE
        bonus REAL;
     BEGIN
        FOR emp_rec IN (SELECT empno, sal, comm FROM emp) LOOP
           bonus := (emp_rec.sal * 0.05) + (emp_rec.comm * 0.25);
           INSERT INTO bonuses VALUES (emp_rec.empno, bonus);
        END LOOP;
        COMMIT;
     END;
end-exec
```

FETCH

The FETCH statement retrieves a row from a cursor.

Format 1

```
EXEC SQL [ AT Database ]

  [ FOR Iterations ]

  FETCH { NEXT                                          }
        { PREVIOUS                                      }
        { PRIOR                                         }
        { FIRST                                         }
        { LAST                                          }
        { NEXT ROWSET                                   }
        { PRIOR ROWSET                                  }
        { FIRST ROWSET                                  }
        { LAST ROWSET                                   }
        { ROWSET STARTING AT { ABSOLUTE } { Host-Var-1 } }
                             { RELATIVE } { Literal-1  }

 FROM { Cursor-Name } [ INTO Host-Variable, ... ]
      { Host-Var-2    }

  END-EXEC
```

Format 2

```
EXEC SQL [ AT Database ]

  FETCH  FROM { Cursor-Name } [ USING DESCRIPTOR { SELDSC         } ]
              { Host-Var    }                    { Sql-Descriptor }
  END-EXEC
```

Syntax rules

1. *Iterations* can be either a host variable or a numeric literal. It specifies the number of rows to be processed.
2. *Cursor-Name* is a Nonnumeric Literal, as defined in the Definitions section of the Preface of this document.
3. *Host-Var-2* must be USAGE HANDLE
4. *Host-Var-1* and *Host-Variable* are host variable.
5. PREVIOUS and PRIOR are synonymous.
6. SELDSC is a group item defined in the Oracle Pro*COBOL SQLDA implementation.
7. *Sql-Descriptor* is a IBM DB2 SQLDA implementation. This syntax is compiled only under the -csdb2 option.

General rules

1. *Cursor-Name* must be previously defined by a DECLARE statement.
2. When Host-Variable is a host variable declared as a group-item, the runtime uses all subordinate items as separate values instead of using the group-item as a single value.
3. If Host-Variable is array, it fetches enough rows to fill the array.
   Array host variables can have different sizes. In this case, the number of rows it fetches is determined by the smaller of the following values:
   - The size of the smallest array
   - The value of the host_integer in the optional FOR clause

If the array is not completely filled then the warning is issued and you should check SQLERRD(3) to see how many rows were actually fetched.

If one of the host variables in the INTO clause is an array, they must all be arrays.

4. When the runtime framework property iscobol.jdbc.cursor.type * is set to its default value 1 (FORWARD_ONLY), moving backward is not allowed.
5. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.
6. Host-Variables are not initialized and they're updated only if the statement is successful. If an error occurs or there are no more records, then the Host-Variables are left unchanged.
7. Positioning of the cursor with rowset-positioned fetch orientations NEXT ROWSET, PRIOR ROWSET, CURRENT ROWSET, and ROWSET STARTING AT RELATIVE is done in relation to the current cursor position. Following a successful rowset-positioned FETCH statement, the cursor is positioned on a rowset of data. The number of rows in the rowset is determined either explicitly or implicitly. The FOR n ROWS clause in the multiple-row-fetch clause is used to explicitly specify the size of the rowset. Positioning is performed relative to the current row or first row of the current rowset, and the cursor is positioned on all rows of the rowset.
A rowset-positioned fetch orientation must not be specified if the current cursor position is not defined to access rowsets. NEXT ROWSET is the only rowset-positioned fetch orientation that can be specified for cursors that are defined as NO SCROLL.
8. ROWSET STARTING AT positions the cursor on the rowset beginning at the row of the result table that is indicated by the ABSOLUTE or RELATIVE specification, and returns data if a target is specified.

Examples

Format 1 - Declare, open and fetch a cursor

```
exec sql
   declare cust_cur  cursor for select * from customers
end-exec

exec sql
   open cust_cur
end-exec

perform until 1 = 2
   exec
      sql fetch next cust_cur into :ws-cust-code, :ws-cust-name
   end-exec

   display "code: " ws-cust-code " name: " ws-cust-name
   if sqlcode = 100
      exit perform
   end-if

end-perform

exec sql
   close cust_cur
end-exec
```

FREE

The FREE statement releases the memory associated to a cursor or prepared statement.

General format

```
EXEC SQL [ AT Database ]

  FREE Host-Variable

END-EXEC
```

Syntax rules

1. *Host-Variable* must be defined as USAGE HANDLE in DATA DIVISION.

General Rules

1. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.

Examples

This sample allocates a cursor handle, uses it and releases it at the end. Allocate is an Oracle specific SQL statement.

```
exec sql allocate :cur-hndl end-exec.

exec sql execute
   begin
        open :cur-hndl for select * from emp;
   end;
end-exec.

exec sql fetch :cur-hndl into :empno,:ename
end-exec.

exec sql free :cur-hndl
end-exec.
```

Prepare a statement to insert a row, execute it and free the handle.

```
working-storage section.
...
77 cmd handle.
...
procedure division.
...
exec sql
     prepare :cmd from
        "insert into cust_table values (2010,'Evan Raymond','New York')"
end-exec

exec sql
     execute :cmd
end-exec

exec sql
     free :cmd
end-exec
```

GRANT

The GRANT statement allows you to manage users and permissions.

General format

```
EXEC SQL [ AT Database ]

  GRANT Options

END-EXEC
```

Syntax rules

1. *Options* is passed to the driver without further checks. Refer to the database documentation for detailed syntax. Syntax errors, if any, are returned at runtime.

General Rules

1. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.

Examples

Grant Select and Insert access privilege to a table to some users

```
exec sql
     grant select, insert on invoices
        to  usr1, erollands
end-exec
```

INCLUDE

The INCLUDE statement includes a file in the source code.

General format

```
EXEC SQL

  INCLUDE File-Name

END-EXEC
```

Syntax rules

1. *File-Name* is the name of a regular disk file. If the name contains a period (.), it must be enclosed in double quotes.

General rules

1. The INCLUDE statement is similar to the COPY statement.
2. File-Name is internally converted to upper case.
3. If there is more than one *File-Name*, only the first one is used, the others are considered comments.

Examples

Include isCOBOL SQLCA copybook

```
exec sql include sqlca end-exec.
```

INSERT

The INSERT statement allows you to add new rows into a table.

General format

```
EXEC SQL [ AT Database ]

  [ FOR Iterations ]

  INSERT Options [ VALUES ( { Insert-Value } ... ) ]

         [ RETURNING Field ... INTO HostVariable ... ]

  END-EXEC
```

Syntax rules

1. *Iterations* can be either a host variable or a numeric literal. It specifies the number of rows to be processed.
2. *Options* is passed to the driver without further checks. Refer to the database documentation for detailed syntax. Syntax errors, if any, are returned at runtime.
3. *Insert-Value* can be a host variable or a Nonnumeric Literal, as defined in the Definitions section of the Preface of this document.
4. *Field* is a alphanumeric literal.
5. *Host-Variable* is a host variable.

General rules

1. When *Insert-Value* is a host variable declared as a group-item, the runtime uses all subordinate items as separate values instead of using the group-item as a single value.
2. The FOR clause limits the number of times the statement is executed when *Insert-Value* are array host variables. If you omit this clause,it executes the statement once for each component of the smallest array.
3. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.
4. The RETURNING clause allows you to receive the updated fields value into host variables. This clause is currently supported only on the Oracle database and requires the -csora compiler option.

Examples

Inserting rows on a customers table

```
exec sql insert into customers values (10, 'Adam Smith') end-exec
exec sql insert into customers values (20, 'Eve Scott') end-exec
```

LOCK TABLE

The LOCK TABLE statement locks a table in a specified mode. This lock manually overrides automatic locking and permits or denies access to a table or view by other users for the duration of your operation.

General format

```
EXEC SQL [ AT Database ]

  LOCK TABLE Options

  END-EXEC
```

Syntax rules

1. *Options* is passed to the driver without further checks. Refer to the database documentation for detailed syntax. Syntax errors, if any, are returned at runtime.

General rules

1. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.

Examples

Lock customers table on Oracle database

```
exec sql lock table customers in row exclusive mode nowait end-exec
```

OPEN

The OPEN statement executes the SELECT statement the cursor refers to and positions the cursor immediately before the first row returned.

Format 1

```
EXEC SQL [ AT Database ]

  OPEN { Cursor-Name } [ USING Host-Variable, ... ] [ INTO Host-Variable, ... ]
       { Host-Var    }

END-EXEC
```

Format 2

```
EXEC SQL [ AT Database ]

  OPEN { Cursor-Name } [ USING DESCRIPTOR Sql-Descriptor ]
       { Host-Var    }

END-EXEC
```

Syntax rules

1. *Cursor-Name* is a Nonnumeric Literal, as defined in the Definitions section of the Preface of this document.
2. *Host-Var* must be USAGE HANDLE
3. *Host-Variable* is a host variable.
4. *Sql-Descriptor* is a SQLDA structure.This syntax is compiled only under the -csdb2 option.

General rules

1. *Cursor-Name* must be previously defined by a DECLARE statement.

Examples

Declare, open and fetch a cursor

```
exec sql
   declare cust_cur  cursor for select * from customers
end-exec

exec sql
   open cust_cur
end-exec

perform until 1 = 2
   exec
      sql fetch next cust_cur into :ws-cust-code, :ws-cust-name
   end-exec

   display "code: " ws-cust-code " name: " ws-cust-name
   if sqlcode = 100
      exit perform
   end-if

end-perform

exec sql
   close cust_cur
end-exec
```

PREPARE

The PREPARE statement encodes a dynamic SQL statement string and assigns it a name.

General format

```
EXEC SQL [ AT Database ]

  PREPARE { Prepared-Statement } [ INTO Sql-Descriptor ] FROM Sql-String
          { Host-Var }

END-EXEC
```

Syntax rules

1. *Prepared-Statement* is a Nonnumeric Literal, as defined in the Definitions section of the Preface of this document.
2. *Host-Var* must be USAGE HANDLE
3. *Sql-String* is a host variable or a Nonnumeric Literal, as defined in the Definitions section of the Preface of this document.
4. *Sql-Descriptor* is a IBM DB2 SQLDA implementation . This syntax is compiled only under the -csdb2 option.

General rules

1. *Sql-String* may contain question marks (?) that will be replaced by constant values or host variables when the statement is executed using the USING clause.
2. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.
3. *Host-Var* should be released throgh a FREE statement before reusing it to store another prepared statement.

Examples

Prepare 2 statements, one to insert a row and one to count how many records meet a criteria

```
*> insert_rec and count_recs do not need to be defined prior to
*> their use in the prepare statements
*> min-key and the-count could be pic 9(4) each

exec sql
     prepare insert_rec from
        "insert into cust_table values (2010,'Evan Raymond','New York')"
end-exec

exec sql
     execute insert_rec
end-exec

exec sql
     prepare count_recs from
        "select count(*) from cust_table where cust_code > ?"
end-exec
move 1990 to min-key
exec sql
     execute count_recs using :min-key
             into :the-count
end-exec

display "Count of records with key > " min-key " : " the-count
```

REVOKE

The REVOKE statement allows you to remove users and permissions.

General format

```
EXEC SQL [ AT Database ]

  REVOKE Options

END-EXEC
```

Syntax rules

1. *Options* is passed to the driver without further checks. Refer to the database documentation for detailed syntax. Syntax errors, if any, are returned at runtime.

General Rules

1. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.

Examples

Revoke Update and Delete access privilege to a table from some users

```
exec sql
     revoke update, delete on invoices
       from usr1, erollands
end-exec
```

ROLLBACK

The ROLLBACK statement aborts the current transaction and automatically starts a new transaction.

General format

```
EXEC SQL [ AT Database ]

  ROLLBACK { WORK        } [ RELEASE ]
           { TRAN        }
           { TRANSACTION }
END-EXEC
```

General Rules

1. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.
2. WORK, TRAN and TRANSACTION are synonymous and are assumed if omitted.
3. The RELEASE option frees all resources (locks and cursors) held by the program and logs off the database.
4. The iscobol.jdbc.autocommit (boolean) configuration setting must be set to false, otherwise all statements are automatically commited. With iscobol.jdbc.autocommit (boolean) set to false, a transaction is started as soon as you connect to the database. The ROLLBACK statement allows you to abort that transaction and

immediately start a new one.

Examples

Rollback transaction

```
exec sql rollback work end-exec
```

SELECT

The SELECT statement returns values from tables.

General format

```
EXEC SQL [ AT Database ]

  [ FOR Iterations ]

  SELECT Select-Options [ INTO Host-Variable, ... ] FROM From-Options
                        [ WHERE Search-Condition  ]  [ Options ]

END-EXEC
```

Syntax rules

1. *Iterations* can be either a host variable or a numeric literal. It specifies the number of rows to be processed.
2. Host variables used in *Search-Condition* and *Select-Options*, if any, are replaced by question marks (?) and then passed to the driver without further check.
3. *Host-Variable* is a host variable.

General rules

1. When *WHERE* is specified, only rows matching *Search-Condition* are returned.
2. When *Search-Condition* is not specified, all rows are returned.
3. The FOR clause limits the number of times the statement is executed when *Search-Condition* contains array host variables If you omit this clause,it executes the statement once for each component of the smallest array.
4. When *Host-Variable* is a host variable declared as a group-item, the runtime uses all subordinate items as separate values instead of using the group-item as a single value.
5. If *Host-Variable* is array, it fetches enough rows to fill the array.
   Array host variables can have different sizes. In this case, the number of rows it fetches is determined by the smaller of the following values:
   - The size of the smallest array
   - The value of the host_integer in the optional FOR clause

   If the array is not completely filled then the warning is issued and you should check SQLERRD(3) to see how many rows were actually fetched.
   If one of the host variables in the INTO clause is an array, they must all be arrays.
6. *Options* are passed to the driver without further checks. Refer to the database documentation for allowed syntax. Syntax errors, if any, are returned at runtime.
7. *Database* identifies the active connection that will execute the query and must be previously defined using a

Format 4 DECLARE statement.

Examples

Select customer name for a given customer code

```
move spaces to ws-cust-name
move 2020   to ws-cust-code
exec sql
     select customer_name into :ws-cust-name
       from customers
      where customer_code = :ws-cust-code
end-exec
```

SET

The SET statement allows you to set specific database dependent settings host variables.

Format 1

```
EXEC SQL [ AT Database ]

  SET Options

END-EXEC
```

Format 2

```
EXEC SQL [ AT Database ]

  SET Host-Variable = Expression

END-EXEC
```

Syntax rules

1. No syntax check is performed on the content of *Options*.
2. *Host-Variable* is a host variable.
3. *Expression* specifies a value and can take a number of different forms. It can be a constant value, a special register, an arithmetic calculation, a function, etcetera...

General rules

1. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.

Format 1

2. The SET statement is passed as is to the underlying JDBC driver. If the driver doesn't understand it, an SQL error will be generated.

Format 2

3. A Format 2 SET statement is automatically transformed to a SELECT query to retrieve the result of the function. The resulting query depends on the compatibility activated through compiler options.

When using the -csdb2 option, the query is:

```
select Expression into :Host-Variable from sysibm.sysdumm1
```

When using the -csora option, the query is:

```
select Expression into :Host-Variable from dual
```

If none of the above options is used, the query is:

```
select Expression into :Host-Variable
```

Examples

Set the current transaction as READ ONLY and give it a name. This syntax is supported by the Oracle database

```
   exec sql
     set transaction read only name 'mytran'
   end-exec
```

Retrieve the current schema through the appropriate special register. This syntax is supported by the DB2 database and requires the -csdb2 compiler option

```
   exec sql
     set :wrk-schema = current schema
   end-exec
```

SET CONNECTION

The SET CONNECTION statement activates a specified connection.

General format

```
EXEC SQL [ AT Database ]

  SET CONNECTION { Connection-name }

END-EXEC
```

Syntax rules

1. *Connection-Name* is a host variable or Nonnumeric Literal, as defined in the Definitions section of the Preface of this document.

General rules

1. When two or more connections are opened you can choose the active connection using the *Connection-Name* parameter.
2. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.

Examples

Switch from one connection to another when having one connection to an Oracle DB and other to a MySQL DB

```
if db-to-use = ora-db
   exec sql
     set connection ora-conn-1
   end-exec
else
   exec sql
     set connection mysql-conn-2
   end-exec
end-if
```

TRUNCATE TABLE

The TRUNCATE TABLE statement deletes all rows in a table.

General format

```
EXEC SQL [ AT Database ]

  TRUNCATE TABLE Table-Name [ Options ]

END-EXEC
```

Syntax rules

1. *Options* is passed to the driver without further checks. Refer to the database documentation for detailed syntax. Syntax errors, if any, are returned at runtime.
2. *Table-Name* is a User-defined word, as defined in the Definitions section of the Preface of this document.

Examples

Empty the table log

```
exec sql
     truncate table log
end-exec
```

UPDATE

The UPDATE statement replaces the values of the specified columns with the values of the specified expressions for all rows of the table that satisfy the search condition.

General format

```
EXEC SQL [ AT Database ]

  [ FOR Iterations ]

  UPDATE Options SET { Column-Name = Column-Value } ...
         [ WHERE { Search-Condition                                    } ]
                 { CURRENT OF Cursor-Name [ FOR ROW Row-Num OF ROWSET ] }

         [ RETURNING Field ... INTO HostVariable ... ]

END-EXEC
```

Syntax rules

1. *Iterations* can be either a host variable or a numeric literal. It specifies the number of rows to be processed.
2. *Options* is passed to the driver without further checks. Refer to the database documentation for detailed syntax. Syntax errors, if any, are returned at runtime.
3. *Column-Name* is a User-defined word, as defined in the Definitions section of the Preface of this document.
4. *Column-Value* is a host variable or a Nonnumeric Literal, as defined in the Definitions section of the Preface of this document.
5. *Field* is an alphanumeric literal.
6. *Row-Num* is a numeric literal or host variable.
7. *Host-Variable* is a host variable.

General rules

1. When *Search-Condition* is specified, only values in rows matching it are replaced.
2. The FOR clause limits the number of times the statement is executed when *Search-Condition* contains array host variables If you omit this clause,it executes the statement once for each component of the smallest array.
3. When *Search-Condition* is not specified, values of all rows are replaced.
4. *Database* identifies the active connection that will execute the query and must be previously defined using a Format 4 DECLARE statement.
5. CURRENT OF allows you to update the last fetched record in the active cursor.
   When this syntax is used
   - the statement associated to the cursor should not use "SELECT *" and
   - the statement associated to the cursor should query from a single table.

   Otherwise a "cursor is not updatable" error may occur.
6. FOR ROW n OF ROWSET specifies which row of the current rowset is to be updated. The corresponding row of the rowset is updated, and the cursor remains positioned on the current rowset.
7. The RETURNING clause allows you to receive the updated fields value into host variables. This clause is currently supported only on the Oracle database and requires the -csora compiler option.

Examples

Apply a discount to customers meeting a criteria

```
exec sql
     update customers
        set discount = 0.10
      where debt_amount < (credit_line * 0.25)
end-exec
```

Declare, open and fetch a cursor. Update records where the field cust-name is empty.

```
exec sql
   declare cust_cur  cursor for select cust_code, cust_name from customers
end-exec

exec sql
   open cust_cur
end-exec

perform until 1 = 2
   exec
      sql fetch next cust_cur into :ws-cust-code, :ws-cust-name
   end-exec

   if ws-cust-name = space
      exec sql
           update customers set cust_name = 'John' where current of cust_cur
      end-exec
   end-if

   if sqlcode = 100
      exit perform
   end-if

end-perform

exec sql
   close cust_cur
end-exec
```

VAR

The VAR statement allows you to mark a host variable as storage for binary data.

General format

```
EXEC SQL

  VAR Host-Variable IS { BINARY   }
                       { LONG RAW }
                       { RAW      }

END-EXEC
```

Syntax rules

1. *Host-Variable* is a host variable.
2. BINARY, LONG RAW and RAW are synonymous.
3. Any value other than BINARY, LONG RAW and RAW is treated as a comment and doesn't have any effect.

General rules

1. The VAR statement must appear in the source after the definition of *Host-Variable*.

Examples

Mark the wrk-ascii data item as RAW:

```
working-storage section.
...
77 wrk-ascii pic x(128).
exec sql var wrk-ascii is raw end-exec.
```

WHENEVER

The WHENEVER statement allows the handling of error and exception conditions.

General format

```
EXEC SQL

  WHENEVER { SQLERROR   } [ DO ] { GO TO Host-Label   }
           { SQLWARNING }        { CALL Host-Label    }
           { NOT FOUND  }        { PERFORM Host-Label}
                                 { CONTINUE           }

END-EXEC
```

Syntax rules

1. *Host-Label* is a User-defined word, as defined in the Definitions section of the Preface of this document.

General rules

1. When SQLERROR is specified, the specified action is performed when the SQL statement generates an error.
2. When SQLWARNING is specified, the specified action is performed when the SQL statement generates a warning.
3. When NOT FOUND is specified, the specified action is performed when the SQL statement affects no rows.
4. When GO TO is specified, the control is transferred to the COBOL paragraph or section identified by *Host-Label*.
5. When CONTINUE is specified, the program continues to the next statement. If a fatal error occurs, an error condition is generated and the program aborts.

Examples

Set the general SQL error procedure

```
main.
   exec sql
        whenever sqlerror do perform test-sql-status
   end-exec
   ...

test-sql-status.
   if sqlcode not = 0
      display message sqlcode   x"0d0a"
                      sqlerrmc
                      title "SQL Error"
      exec sql disconnect all end-exec
      goback
   end-if
   .
```

Chapter 13

EFD Directives

EFD directives are special comments that allow the customization of the generation of the EFD file when the -efd or -efc compiler options are used. There are three syntaxes to specify an EFD directive.

Syntax 1:

```
>>EFD directive-name [=directive-value]
```

Syntax 2:

```
$EFD directive-name [=directive-value]
```

Syntax 3:

```
*(( EFD directive-name [=directive-value] ))
```

The >> marker can appeear everywhere except the sequence number area (columns 1 to 6) of the ANSI source format.

The $ and the * markers must be in the comment area, that is column 7 in ANSI source format and column 1 in Terminal source format.

If the source code is written in Free format, EFD directives can appear at any column with the following syntax.

Syntax 4:

```
*>(( EFD directive-name [=directive-value] ))
```

The directive must appear above the field or the file that you want to customize in the FILE SECTION of the source code.

Note that no spaces are allowed between the comment symbol and the couple of parenthesis. If spaces are present, the directive is treated as a standard comment and doesn't have effects.

For compatibility with Acucobol-GT, the XFD keyword is supported. $XFD is synonymous with $EFD, while *((XFD is synonymous with *((EFD.

EFD directives are ignored in Object and Factory methods. They can be used only in standard programs.

Defining multiple EFD directives in the same point

Multiple EFD directives can be distributed on multiple lines as follows

Syntax 1:

```
>>EFD ALPHA
>>EFD NAME=field1
 03 fd1-field1 pic 9(5).
```

Syntax 2:

```
$EFD ALPHA
$EFD NAME=field1
 03 fd1-field1 pic 9(5).
```

Syntax 3:

```
*((EFD ALPHA))
*((EFD NAME=field1))
 03 fd1-field1 pic 9(5).
```

Syntax 4:

```
*>((EFD ALPHA))
*>((EFD NAME=field1))
 03 fd1-field1 pic 9(5).
```

or merged in a single line as follows:

Syntax1:

```
>>EFD ALPHA, NAME=field1
 03 fd1-field1 pic 9(5).
```

Syntax 2:

```
$EFD ALPHA, NAME=field1
 03 fd1-field1 pic 9(5).
```

Syntax 3:

```
*((EFD ALPHA, NAME=field1))
 03 fd1-field1 pic 9(5).
```

Syntax 4:

```
*>((EFD ALPHA, NAME=field1))
 03 fd1-field1 pic 9(5).
```

Splitting a EFD directive on multiple lines

A single EFD directive can be splitted on multiple lines by repeating the EFD delimiters on each line.

For example, the following snippets

Syntax 1:

```
>>EFD NAME=
>>EFD field1
 03 fd1-field1 pic 9(5).
```

Syntax 2:

```
$EFD NAME=
$EFD field1
 03 fd1-field1 pic 9(5).
```

Syntax 3:

```
*((EFD NAME=))
*((EFD field1 ))
 03 fd1-field1 pic 9(5).
```

Syntax 4:

```
*>((EFD NAME=))
*>((EFD field1 ))
 03 fd1-field1 pic 9(5).
```

are equivalent to

Syntax 1:

```
>>EFD NAME= field1
 03 fd1-field1 pic 9(5).
```

Syntax 2:

```
$EFD NAME= field1
 03 fd1-field1 pic 9(5).
```

Syntax 3:

```
*((EFD NAME= EFD field1 ))
 03 fd1-field1 pic 9(5).
```

Syntax 4:

```
*>(($EFD NAME= field1 ))
 03 fd1-field1 pic 9(5).
```

ALPHA Directive

In order to store non-numeric data (like, LOW-VALUES or special codes) in numeric keys, this directive allows a data item that has been defined as numeric in the COBOL program to be treated as alphanumeric text (CHAR (n) n 1-max column length) in the database.

```
>>EFD ALPHA
```

or

```
$EFD ALPHA
```

or

```
*(( EFD ALPHA ))
```

or

```
*>(( EFD ALPHA ))
```

Moving a non-numeric value such as "A234" into numeric fields without using the $EFD ALPHA directive will cause the field content to be treated as invalid data.

Example

In the following example, the KEY IS code-key has been specified and the following record definition is present. CODE-NUM is a numeric value and is the key field, since group items are disregarded in the database.

```
01 EMPLOYEE-RECORD.
    05 EMP-KEY.
       10 EMP-NUM           PIC 9(5).
```

Using the $EFD ALPHA directive will change a non-numeric value such as "A234" so that the record will not be rejected by the database, since "A234" is an alphanumeric value and CODE-NUM is a numeric value.

```
 01 EMPLOYEE-RECORD.
    05 EMP-KEY.
>>EFD ALPHA
       10 EMP-NUM           PIC 9(5).
```

Now, the following operation can be used without worrying about rejection.

```
MOVE "C0531" TO CODE-KEY.
WRITE CODE-RECORD.
```

ALWAYS Directive

The WHEN Directive distributes fields into separate tables depending on conditions known as "WHEN conditions". When a file description contains WHEN Directives, some records may be skipped because they don't meet any WHEN conditions. The ALWAYS directive makes all records available in a named table regardless of whether they meet WHEN conditions. This directive should be specified at the record level.

This directive only affects iss dictionaries and therefore it can be used only along with the -efc compiler option.

```
>>EFD ALWAYS
```

or

```
$EFD ALWAYS
```

or

```
*(( EFD ALWAYS ))
```

or

```
*>(( EFD ALWAYS ))
```

Example

The following FD:

```
       FD tab1.
      >>EFD ALWAYS TABLENAME=MMTAB1_ALL
       01 rec.
           02 rec-KEY pic x(12).
           02 field1  pic x.
      >>EFD WHEN FIELD1 = 1 TABLENAME=MMTAB2
           02 field2  pic x.
      >>EFD WHEN FIELD1 = 2 TABLENAME=MMTAB3
           02 field3  pic x.
           02 field4  pic x.
```

Generates the following tables on c-tree SQL.

The record with FIELD1=3 would be lost because it doesn't match any of the $EFD WHEN directives, but thanks to the $EFD ALWAYS directive, it can be stored in a table where all the records are shown, regardless of the value of FIELD1.

BINARY Directive

This directive is used to specify that the data in the field could be alphanumeric data of any classification. Absolutely any data is allowed. The method of storing fields declared as binary is database-specific. For example, Oracle uses the data type RAW for the field, while PostgreSQL uses BYTEA.

You might use this directive when you need to store a key that contains LOW or HIGH values; COBOL allows a numeric field to contain LOW or HIGH values, but these are invalid for a numeric field in a database.

```
>>EFD BINARY
```

or

```
$EFD BINARY
```

or

```
*(( EFD BINARY ))
```

or

```
*>(( EFD BINARY ))
```

Example

The field EMP-NUM will apear as a binary field in the database:

```
 01 EMPLOYEE-RECORD.
    05 EMP-KEY.
>>EFD BINARY
       10 EMP-NUM           PIC 9(5).
```

BINDEFAULT Directive

This directive affects only iss files and not EFD files, therefore it can be used only with c-tree SQL. It allows to specify the hex initial value for a specific field.

```
>>EFD BINDEFAULT=DefaultValue
```

or

```
$EFD BINDEFAULT=DefaultValue
```

or

```
*(( EFD BINDEFAULT=DefaultValue ))
```

or

```
*>(( EFD BINDEFAULT=DefaultValue ))
```

DefaultValue is generated in the iss file as is, without any conversion

This directive is useful for setting hex values as initial data for FD fields, otherwise the CBDEFAULT Directive should be used.

Example

The following setting initializes field-1 to low-values.

```
>>EFD BINDEFAULT='\x00\x00'
 03 field-1 PIC X(32).
```

CBDEFAULT Directive

This directive affects only iss files and not EFD files, therefore it can only be used with c-tree SQL. It allows the specification of the initial value for a specific field. FD fields are automatically initialized to space if they're alphanumeric and to zero if they're numeric or dates. The default initialization can be changed by using the -dv compiler option. If you need a specific field of the FD to be initialized to a particular value, you can take advantage of the $EFD CBDEFAULT directive. If CBDEFAULT and BINDEFAULT are set on the same field, CBDEFAULT will have priority.

```
>>EFD CBDEFAULT=DefaultValue
```

or

```
$EFD CBDEFAULT=DefaultValue
```

or

```
*(( EFD CBDEFAULT=DefaultValue ))
```

or

```
*>(( EFD CBDEFAULT=DefaultValue ))
```

DefaultValue is generated in the iss file as is, without any conversion.

Example

The following setting initializes field-1 to “A” .

```
>>EFD CBDEFAULT='A'
 03 field-1 PIC X.
```

COBTRIGGER Directive

This directive allows for the definition of a COBOL program as a trigger of I/O events such as READ, WRITE, REWRITE, or DELETE. This defined COBOL program is automatically called before and after every I/O event. It must appear above the file definition in the FILE-CONTROL paragraph.

```
>>EFD COBTRIGGER=ProgramName
```

or

```
$EFD COBTRIGGER=ProgramName
```

or

```
*(( EFD COBTRIGGER=ProgramName ))
```

or

```
*>(( EFD COBTRIGGER=ProgramName ))
```

Example

The program MYTRIGGER will be called for each operation on MYTABLE:

```
>>EFD COBTRIGGER=MYTRIGGER
 SELECT MYTABLE ASSIGN TO "MYTABLE"
        ORGANIZATION INDEXED
        ACCESS DYNAMIC
        RECORD KEY MY-KEY.
```

This directive is supported for compatibility with third party interfaces. isCOBOL DatabaseBridge and c-tree SQL do not currently support it.

COMMENT Directive

This directive is used to include comments in an EFD file. In this way, information can be embedded in an EFD file so that other applications can access the data dictionary.

```
>>EFD COMMENT FreeText
```

or

```
$EFD COMMENT FreeText
```

or

```
*(( EFD COMMENT FreeText ))
```

or

```
*>(( EFD COMMENT FreeText ))
```

Example

Put a comment over a field

```
>>EFD COMMENT This field will be removed in the future
 03 field-1 PIC X.
```

DATE Directive

DATE, TIME and TIMESTAMP are special field types that are not supported natively by COBOL. To use such fields, a conversion is necessary. The DATE directive defines the conversion rules.

```
>>EFD DATE [ = { DateFormatString       }  ]
               { TimeFormatString       }
               { DateTimeFormatString   }
               { JulianDateFormatString }
```

or

```
$EFD DATE [ = { DateFormatString       }  ]
              { TimeFormatString       }
              { DateTimeFormatString   }
              { JulianDateFormatString }
```

or

```
*(( EFD DATE [ = { DateFormatString       }  ] ))
                 { TimeFormatString       }
                 { DateTimeFormatString   }
                 { JulianDateFormatString }
```

or

```
*>(( EFD DATE [ = { DateFormatString       }  ] ))
                  { TimeFormatString       }
                  { DateTimeFormatString   }
                  { JulianDateFormatString }
```

If *DateFormatString* is not specified, then six-digit (or six-character) fields are retrieved as YYMMDD from the database, while eight-digit fields are retrieved as YYYYMMDD.

Format strings are the representation of the format, composed of a sequence of characters. Each character represents a digit of the COBOL field. Possible values for characters are:

Character	Meaning
J	Julian date digit
E	Day of year (001–366)
Y	Year (2 or 4 digit)
M	Month (01-12)
D	Day of month (01-31)
H	Hour (00-23)
N	Minute (00-59)
S	Second (00-59)
T	Cents (00-99)

The following table lists all supported *DateFormatStrings*:

DateFormatString	Meaning
YYMMDD	big endian date with two digit year
YYYYMMDD	big endian date with four digit year
MMDDYY	middle endian date with two digit year
MMDDYYYY	middle endian date with four digit year
DDMMYY	little endian date with two digit year
DDMMYYYY	little endian date with four digit year

DateFormatString	Meaning
YYYYEEE	four digit year followed by day of year
EEEYYYY	day of year followed by four digit year
YYEEE	two digit year followed by day of year
EEEYY	day of year followed by two digit year

The following table lists all supported *TimeFormatStrings*:

TimeFormatString	Meaning
HH	hours
HHNN	hours and minutes
HHNNSS	hours, minutes and seconds
HHNNSSTT	hours, minutes, seconds and cents

DateFormatStrings and *TimeFormatStrings* from the above tables can be combined together into a *DateTimeFormatString*. There are two types of *DateTimeFormatStrings*:

1. DateFormatString + TimeFormatString
2. TimeFormatString + DateFormatString

JulianDateFormatString is a serie of "J" characters. Up to eight "J" characaters are allowed. Only for iss dictionaries it is possible to specify the julian base date by setting the configuration property iscobol.compiler.iss_julian_base at compile time.

When the COBOL field contains more digits than the ones required to store the value, leading digits are set to 0. When the COBOL field contains less digits than the ones required to store the value, leading digits are truncated. When the date or time values are not complete (e.g.: a time field with hours and minutes only), they're completed with default values by the database.

You may place the DATE directive in front of a group item, so long as you also use the USE GROUP directive.

Note: c-tree SQL requires that the number of digits in the format string matches with the number of digits of the data-item, otherwise a conversion error occurs. In addition, c-tree SQL doesn't support hundredths of seconds (the value T in the format string), therefore they're truncated.

Example

The next snippets show how to define the DATE-PURCHASED field as a date on the database. Note that when the field is a group item, the USE GROUP directive is used as well.

```
>>EFD DATE=YYYYMMDD
 05 DATE-PURCHASED      PIC 9(08).
 05 PAY-METHOD          PIC X(05).
```

The column DATE_PURCHASED will have eight digits and will be type DATE in the database, with a format of YYYYMMDD.

```
>>EFD DATE=YYYYMMDD, USE GROUP
 05 DATE-PURCHASED.
    10 YYYY              PIC 9(04).
    10 MM                PIC 9(02).
    10 DD                PIC 9(02).
 05 PAY-METHOD           PIC X(05).
```

FILE Directive

The FILE directive names the data dictionary with the file extension .xml. This directive is required when creating a different EFD name from that specified in the SELECT COBOL statement. It is also required when the COBOL file name is not specific.

```
>>EFD FILE=FileName
```

or

```
$EFD FILE=FileName
```

or

```
*(( EFD FILE=FileName ))
```

or

```
*>(( EFD FILE=FileName ))
```

Example

In this case, the isCOBOL compiler makes an EFD file name called CUSTOMER.XML.

```
 ENVIRONMENT DIVISION.
 FILE-CONTROL.
 SELECT FILENAME ASSIGN TO VARIABLE-OF-WORKING.
 ...
 DATA DIVISION.
 FILE SECTION.
>>EFD FILE=CUSTOMER
 FD FILENAME
```

FIX-LENGTH Directive

This directive forces an alphanumeric COBOL field to be defined as CHAR on the database.

```
>>EFD FIX-LENGTH
```

or

```
$EFD FIX-LENGTH
```

or

```
*(( EFD FIX-LENGTH ))
```

or

```
*>(( EFD FIX-LENGTH ))
```

Example

```
>>EFD FIX-LENGTH
 03 CUS-NAME PIC X(32).
```

HIDDEN Directive

This directive allows a field to be marked as hidden. This will avoid the need to convert a field into a database field. The field will be ignored by the external interfaces that read the EFD file.

```
>>EFD HIDDEN
```

or

```
$EFD HIDDEN
```

or

```
*(( EFD HIDDEN ))
```

or

```
*>(( EFD HIDDEN ))
```

Example

The field USELESS will not appear in the database

```
>>EFD HIDDEN
 05 USELESS        PIC X(32).
```

HINT Directive

The HINT directive allows the specification of a custom hint that will be used for a specific key when working on Oracle database with DatabaseBridge. If specified, this hint is used instead of the hint that is automatically generated by -oh option.

```
>>EFD HINT idxNum:hintValue
```

or

```
$EFD HINT idxNum:hintValue
```

or

```
*(( EFD HINT idxNum:hintValue ))
```

or

```
*>(( EFD HINT idxNum:hintValue ))
```

idxNum is the number of the key that will be affected by the hint. This number is calculated considering the order of the keys in the File-Control, where 1 is the primary key while 2 and higher values are the alternate keys.

hintValue can be a string or a number.

- If a string is used, the string is generated as it is in the SQL hint. For example, the following directive generates the hint */*+ *ordered* */* for readings on the first key.

```
 FD file1.
$EFD HINT 1:"ordered"
 01 file1-record.
 ...
```

- If a number is used, the number specifies the index to be suggested for the query in the SQL hint. For example, the directive below generates a hint that makes Oracle use the first index when the COBOL program is reading the second key of the file:

```
 FD file1.
$EFD HINT 2:1
 01 file1-record.
 ...
```

NAME Directive

The NAME directive assigns an RDBMS column name to the field defined on the next line. This directive can be used to avoid problems created by columns with incompatible or duplicate names.

If the name is written between quotes, then the case is preserved, otherwise it's made uppercase.

```
>>EFD NAME=ColumnName
```

or

```
$EFD NAME=ColumnName
```

or

```
*(( EFD NAME=ColumnName ))
```

or

```
*>(( EFD NAME=ColumnName ))
```

Example

In the RDBMS, the COBOL field cus-cod will map to an RDBMS field named CUSTOMERCODE.

```
>>EFD NAME=CUSTOMERCODE
 05 CUS-COD         PIC 9(05).
```

NOCONVERTERROR Directive

The NOCONVERTERROR directive is deprecated and supported only for backward compatibility because NOCONVERTERROR is now applied by default to all the record definitions.

The NOCONVERTERROR directive instructs the c-treeSQL to return NULL instead of raising an error where a conversion error (such as spaces or zeros in a date field, for example) occurs on the record.

It affects the next record definition so, if the FD contains more than one record definition and you want to avoid conversion errors on the whole file, you should place the directive above each record definition.

```
>>EFD NOCONVERTERROR
```

or

```
$EFD NOCONVERTERROR
```

or

```
*(( EFD NOCONVERTERROR ))
```

or

```
*>(( EFD NOCONVERTERROR ))
```

Example

No conversion errors will be shown for file1-record.

```
  FD FILE1.
>>EFD NOCONVERTERROR
 01 FILE1-RECORD.
    03 FILE1-KEY   PIC 9(5).
```

NUMERIC Directive

The NUMERIC directive causes the subsequent field to be treated as a numeric if it is declared as alphanumeric.

```
>>EFD NUMERIC
```

pr

```
$EFD NUMERIC
```

or

```
*(( EFD NUMERIC ))
```

or

```
*>(( EFD NUMERIC ))
```

Example

The field customer-code will be stored as numeric data type in the table.

```
>>EFD NUMERIC
 03 CUSTOMER-CODE PIC X(7).
```

READ-ONLY Directive

The READ-ONLY directive causes the subsequent field to be marked as read-only in the dictionary files generated by the -efc and -efd compiler options. Currently isCOBOL DatabaseBridge and c-tree SQL ignore this information.

```
>>EFD READ-ONLY
```

pr

```
$EFD READ-ONLY
```

or

```
*(( EFD READ-ONLY ))
```

or

```
*>(( EFD READ-ONLY ))
```

Example

The field customer-code will be marked as read-only in the dictionary file.

```
$EFD READ-ONLY
 03 CUSTOMER-CODE PIC X(7).
```

SERIAL Directive

This directive is used to define a serial data field. To trigger the generation of a serial number by isCOBOL, insert a record and supply 0 value for the serial field. If you insert a new row and supply an integer value instead of a 0 value, isCOBOL will not generate a serial number. If the supplied integer value is greater than the last serial number generated, isCOBOL will reset the sequence of generated serial numbers to start with the supplied integer value.

```
>>EFD SERIAL
```

or

```
$EFD SERIAL
```

or

```
*(( EFD SERIAL ))
```

or

```
*>(( EFD SERIAL ))
```

Example

```
 01 EMPLOYEE-RECORD.
    05 EMP-KEY.
       10 EMP-TYPE PIC X.
>>EFD SERIAL
       10 EMP-COUNT PIC 9(05).
```

This directive is supported for compatibility with third party interfaces. isCOBOL DatabaseBridge and c-tree SQL don't currently support it.

SPLIT Directive

This directive is used to define one or more table splitting points starting where the interface makes a new RDBMS table.

```
>>EFD SPLIT
```

or

```
$EFD SPLIT
```

or

```
*(( EFD SPLIT ))
```

or

```
*>(( EFD SPLIT ))
```

Example

A COBOL FD structure using the SPLIT directive. In this example three tables named INVOICE, INVOICE_A, and INVOICE_B are created with fields between the split points.

```
FILE SECTION.
FD INVOICE.
01 INV-RECORD-TOP.
   03 INV-KEY.
      05 INV-TYPE PIC X.
      05 INV-NUMBER PIC 9(5).
      05 INV-ID PIC 999.
   03 INV-CUSTOMER PIC X(30).
>>EFD SPLIT
   03 INV-KEY-D.
      05 INV-TYPE-D PIC X.
      05 INV-NUMBER-D PIC 9(5).
      05 INV-ID-B PIC 999.
>>EFD SPLIT
   03 INV-ARTICLES PIC X(30).
   03 INV-QTA PIC 9(5).
   03 INV-PRICE PIC 9(17).
```

This directive is supported for compatibility with third party interfaces. isCOBOL DatabaseBridge and c-tree SQL don't currently support it.

TEMPORARY Directive

The TEMPORARY directive instructs the DatabaseBridge to treat the file as a temporary table instead of a physical table.

```
>>EFD TEMPORARY[=GLOBAL|PRIVATE]
```

or

```
$EFD TEMPORARY[=GLOBAL|PRIVATE]
```

or

```
*(( TEMPORARY[=GLOBAL|PRIVATE] ))
```

or

```
*>(( TEMPORARY[=GLOBAL|PRIVATE] ))
```

If TEMPORARY=GLOBAL is used, DatabaseBridge treats the file as a global temporary table.

If TEMPORARY=PRIVATE is used, DatabaseBridge treats the file as a private temporary table. Private temporary tables are supported only by Oracle starting from version 18c. Trying to create a private temporary table on a previous Oracle version generates an error. EDBI routines for databases that are not Oracle always create global temporary tables.

If just TEMPORARY is used, then TEMPORARY=GLOBAL is assumed.

This directive is supported only by DatabaseBridge.

Example

Define a temporary table.

```
      >>EFD TEMPORARY
       FD TEMP-TABLE.
       01 TT-RECORD.
          03 TT-KEY PIC 9(5).
          03 TT-DATA PIC X(10).
```

USE GROUP Directive

The USE GROUP directive assigns a group of items to a single column in the table. The default data type for the resultant dataset in the database column is alphanumeric (CHAR (n), where n=1-max column length). The directive may be combined with other directives if the data is stored as a different type (DATE, NUMERIC). Combining fields into groups improves processing speed on the database, so effort is made to determine which fields can be combined.

```
>>EFD USE GROUP
```

or

```
$EFD USE GROUP
```

or

```
*(( EFD USE GROUP ))
```

or

```
*>(( EFD USE GROUP ))
```

Example

By adding the USE GROUP directive, the data is stored as a single numeric field where the column name is code-key.

```
 01 CODE-RECORD.
>>EFD USE GROUP
    05 CODE-KEY.
       10 AREA-CODE-NUM     PIC 9(03).
       10 CODE-NUM          PIC 9(07).
```

The USE GROUP directive can be combined with other directives. The fields are mapped into a single DATE type data column in the database.

```
>>EFD DATE, USE GROUP
    05 DATE-PURCHASED.
       10 YYYY              PIC 9(04).
       10 MM                PIC 9(02).
       10 DD                PIC 9(02).
```

VAR-LENGTH Directive

This directive forces an alphanumeric COBOL field to be defined as VARCHAR in the database.

```
>>EFD VAR-LENGTH
```

or

```
$EFD VAR-LENGTH
```

or

```
*(( EFD VAR-LENGTH ))
```

or

```
*>(( EFD VAR-LENGTH ))
```

Example

```
>>EFD VAR-LENGTH
  03 CUS-NAME PIC X(32).
```

This directive is not supported by c-tree SQL.

WHEN Directive

The WHEN directive is used to build certain columns in the RDBMS that wouldn't normally be built by default. By specifying a WHEN directive in the code, the field (and subordinate fields in the case of a group item) immediately following this directive will appear as an explicit column, or columns, in the database tables.

The database stores and retrieves all fields regardless of whether they are explicit or not. Furthermore, key fields and fields from the largest record automatically become explicit columns in the database table. The WHEN directive is only used to guarantee that additional fields will become explicit columns when you want to include multiple record definitions.

One condition for how the columns are to be used is specified in the WHEN directive. Additional fields that are to become explicit columns in a database table must not be FILLER or occupy the same area as key fields.

```
>>EFD WHEN Conditions [ TABLENAME = TableName ]
```

or

```
$EFD WHEN Conditions [ TABLENAME = TableName ]
```

or

```
*(( EFD WHEN Conditions [ TABLENAME = TableName ] ))
```

or

```
*>(( EFD WHEN Conditions [ TABLENAME = TableName ] ))
```

Conditions is a sequence of one or more conditions separated by AND and OR operators*:*

```
Condition [ {AND} Condition [...[ {AND} Condition ] ] ]
            {OR }                 {OR }
```

Condition is

```
{ Field { =  } Value }
        { <= }
        { <  }
        { >= }
        { >  }
        { != }
{ Field   =    OTHER }
```

Field is a previously defined COBOL field. This field must be part of the primary key when you work with isCOBOL DatabaseBridge and c-tree SQL features.

Value is an explicit data value enclosed in quotes.

When Field = OTHER is used, the field or fields after OTHER must be used only if the WHEN condition or conditions listed at the same level are not met. OTHER can be used before one record definition, and, within each record definition, once at each level. It is necessary to use a WHEN directive with OTHER in the eventuality that the data in a field doesn't meet the explicit conditions specified in the other WHEN directives, otherwise, the results will be undefined.

When TABLENAME is used, the name of the table is changed at run-time according to the condition defined by the WHEN directive.

When you work with c-tree SQL features, every record definition in a multi-record FD should be mapped with a WHEN directive with TABLENAME.

When multiple conditions are specified in a source code in fixed format, it's possible that the directive extends over columns 72. In this case, consider splitting the directive on multiple lines as explained in Splitting a EFD directive on multiple lines.

Example

A COBOL FD structure using the "When" directive with two table names.

```
      >>EFD FILE=INV
       FD  INVOICE.
      >>EFD WHEN INV-TYPE = "A" TABLENAME = INV-HEADER
       01  INV-RECORD-HEADER.
        03 INV-KEY.
           05 INV-TYPE          PIC X.
           05 INV-NUMBER        PIC 9(5).
           05 INV-ID            PIC 999.
        03 INV-CUSTOMER         PIC X(30).

      >>EFD WHEN INV-TYPE = "B" TABLENAME = INV-DETAILS
       01  INV-RECORD-DETAILS.
        03 INV-KEY-D.
           05 INV-TYPE-D        PIC X.
           05 INV-NUMBER-D      PIC 9(5).
           05 INV-ID-D          PIC 999.
        03 INV-ARTICLES         PIC X(30).
        03 INV-QTA              PIC 9(5).
        03 INV-PRICE            PIC 9(17).
```

The interface uses two tables named "inv_header" and "inv_details" according to the value of INV-TYPE field.

```
      *>WRITE HEADER ROW
           MOVE "A" TO INV-TYPE
           MOVE 1   TO INV-NUMBER
           MOVE 0   TO INV-ID
           MOVE "acme company" TO INV-CUSTOMER
           WRITE INV-RECORD-HEADER
      *>WRITE DETAIL ROWS
           MOVE "B" TO INV-TYPE
           MOVE 1   TO INV-NUMBER
           MOVE 0   TO INV-ID
           MOVE "floppy disk" TO INV-ARTICLES
           MOVE 10  TO INV-QTA
           MOVE 123 TO INV-PRICE
           WRITE INV-RECORD-DETAILS
```

Running the code above with isCOBOL DatabaseBridge, for example, the EDBI routine fills the INV-RECORD-HEADER record in the "inv_header" table and INV-RECORD-DETAILS in the "inv_details" table.

EFD WHEN can also be used with REDEFINES:

```
      >>EFD FILE=INV
       FD  INVOICE.
       01 REC-INVOICE.
          03 INV-KEY.
             05 INV-TYPE          PIC X.
             05 INV-NUMBER        PIC 9(5).
             05 INV-ID            PIC 999.
      >>EFD WHEN INV-TYPE = "A" TABLENAME = INV-HEADER
          03  INV-RECORD-HEADER.
             05 INV-CUSTOMER         PIC X(30).
      >>EFD WHEN INV-TYPE = "B" TABLENAME = INV-DETAILS
          03  INV-RECORD-DETAILS REDEFINES INV-RECORD-HEADER.
              05 INV-ARTICLES      PIC X(30).
              05 INV-QTA           PIC 9(5).
              05 INV-PRICE         PIC 9(17).
```

When TABLENAME is not used, all fields are handled in the same table. In this case, fields under WHEN Directive are filled only when the condition is true.

A COBOL FD structure using the "When" directive without TABLENAME.

```
      >>EFD FILE=CUST
       FD  CUST.
       01  CUST-RECORD.
           03 CUST-COD         PIC X(5).
           03 CUST-NAME1       PIC X(35).
           03 CUST-NAME2       PIC X(35).
           03 CUST-CONTACT     PIC 9. |1 for e-mail, 2 for cell phone
           03 CUST-CELL        PIC X(32).
      >>EFD WHEN CUST-CONTACT = "1"
           03 CUST-EMAIL       PIC X(32) REDEFINES CUST-CELL.
```

WHEN OTHER Directive

The WHEN OTHER directive is a special form of WHEN directive where no specific field is tested. By specifying a WHEN OTHER directive in the code, the field (and subordinate fields in the case of a group item) immediately following this directive will appear as an explicit column, or columns, in the database tables when all the conditions specified by other WHEN directives are false.

This directive is particularly useful when multiple fields are tested by other WHEN directives, making it difficult to create a condition that is satisfied when all the other conditions are false.

This directive can be used only to map a record definition to a RDBMS table, hence the TABLENAME clause is madatory.

```
>>EFD WHEN OTHER  TABLENAME = TableName
```

or

```
$EFD WHEN OTHER  TABLENAME = TableName
```

or

```
*(( EFD WHEN OTHER  TABLENAME = TableName ))
```

or

```
*>(( EFD WHEN OTHER  TABLENAME = TableName ))
```

Example

A COBOL FD structure using the "When" directive with two table names.

```
      >>EFD FILE=INV
       FD  INVOICE.
      >>EFD WHEN INV-TYPE = "H" OR INV-TYPE = "A" TABLENAME = INV-HEADER
       01  INV-RECORD-HEADER.
        03 INV-KEY.
           05 INV-TYPE          PIC X.
           05 INV-NUMBER        PIC 9(5).
           05 INV-ID            PIC 999.
        03 INV-CUSTOMER         PIC X(30).

      >>EFD WHEN OTHER TABLENAME = INV-DETAILS
       01  INV-RECORD-DETAILS.
        03 INV-KEY-D.
           05 INV-TYPE-D        PIC X.
           05 INV-NUMBER-D      PIC 9(5).
           05 INV-ID-D          PIC 999.
        03 INV-ARTICLES         PIC X(30).
        03 INV-QTA              PIC 9(5).
        03 INV-PRICE            PIC 9(17).
```

We assume that every record that is not a header row (it doesn't have neither INV-TYPE="H" nor INV-TYPE="A") is a detail row.

The interface uses two tables named "inv_header" and "inv_details" according to the value of INV-TYPE field.

```
      *>WRITE HEADER ROW
           MOVE "H" TO INV-TYPE
           MOVE 1   TO INV-NUMBER
           MOVE 0   TO INV-ID
           MOVE "acme company" TO INV-CUSTOMER
           WRITE INV-RECORD-HEADER
      *>WRITE DETAIL ROWS
           MOVE "D" TO INV-TYPE
           MOVE 1   TO INV-NUMBER
           MOVE 0   TO INV-ID
           MOVE "floppy disk" TO INV-ARTICLES
           MOVE 10  TO INV-QTA
           MOVE 123 TO INV-PRICE
           WRITE INV-RECORD-DETAILS
```

Running the code above with isCOBOL DatabaseBridge, for example, the EDBI routine fills the INV-RECORD-HEADER record in the "inv_header" table and INV-RECORD-DETAILS in the "inv_details" table.

Chapter 11

ELK Directives

ELK directives are special comments that allow the customization of the generation of the Web Service bridge programs when the iscobol.compiler.servicebridge (boolean) is set to true. There are three syntaxes to specify an ELK directive.

Syntax 1:

```
>>ELK directive-name [=directive-value]
```

Syntax 2:

```
$ELK directive-name [=directive-value]
```

Syntax 3:

```
*(( ELK directive-name [=directive-value] ))
```

The >> marker can appeear everywhere except the sequence number area (columns 1 to 6) of the ANSI source format.

The $ and the * markers must be in the comment area, that is column 7 in ANSI source format and column 1 in Terminal source format.

If the source code is written in Free format, ELK directives can appear at any column with the following syntax.

Syntax 4:

```
*>(( ELK directive-name [=directive-value] ))
```

The directive must appear above the field or the file that you want to customize in the LINKAGE SECTION of the source code.

Note that no spaces are allowed between the comment symbol and the couple of parenthesis. If spaces are present, the directive is treated as a standard comment and doesn't have effects.

ELK directives are ignored in Object and Factory methods. They can be used only in standard programs.

Defining multiple ELK directives in the same point

Multiple ELK directives can be distributed on multiple lines as follows

Syntax 1::

```
>>ELK INPUT
>>ELK TYPE=long
 03 fd1-field1 pic 9(5).
```

Syntax 2:

```
$ELK INPUT
$ELK TYPE=long
 03 fd1-field1 pic 9(5).
```

Syntax 3:

```
*((ELK INPUT))
*((ELK TYPE=long))
 03 fd1-field1 pic 9(5).
```

Syntax 4:

```
*>((ELK INPUT))
*>((ELK TYPE=long))
 03 fd1-field1 pic 9(5).
```

or merged in a single line as follows:

Syntax 1:

```
>>ELK INPUT, TYPE=long
 03 fd1-field1 pic 9(5).
```

Syntax 2:

```
$ELK INPUT, TYPE=long
 03 fd1-field1 pic 9(5).
```

Syntax 3:

```
*((ELK INPUT, TYPE=long))
 03 fd1-field1 pic 9(5).
```

Syntax 4:

```
*>((ELK INPUT, TYPE=long))
 03 fd1-field1 pic 9(5).
```

Splitting a ELK directive on multiple lines

A single ELK directive can be splitted on multiple lines by repeating the ELK delimiters on each line.

For example, the following snippets

Syntax 1

```
>>ELK TYPE=
>>ELK long
 03 fd1-field1 pic 9(5).
```

Syntax 2:

```
$ELK TYPE=
$ELK long
 03 fd1-field1 pic 9(5).
```

Syntax 3:

```
*((ELK TYPE=))
*((ELK long ))
 03 fd1-field1 pic 9(5).
```

Syntax 4:

```
*>(($ELK TYPE=))
*>((ELK long ))
 03 fd1-field1 pic 9(5).
```

are equivalent to

Syntax 1:

```
>>ELK TYPE= long
 03 fd1-field1 pic 9(5).
```

Syntax 2:

```
$ELK TYPE= long
 03 fd1-field1 pic 9(5).
```

Syntax 3:

```
*((ELK TYPE= long ))
 03 fd1-field1 pic 9(5).
```

Syntax 4:

```
*>((ELK TYPE= long ))
 03 fd1-field1 pic 9(5).
```

DECORATION Directive

The DECORATION directive specifies if field names should be decorated or not. By default the decoration is applied, so the Web Service field names will consist in the COBOL field names followed by either "_in" or "_out". If you wish the Web Service field names to match the COBOL field name, use DECORATION=NONE. This directive affects the whole Linkage Section.

```
>>ELK DECORATION=DEFAULT|NONE
```

or

```
$ELK DECORATION=DEFAULT|NONE
```

or

```
*(( ELK DECORATION=DEFAULT|NONE ))
```

or

```
*>(( ELK DECORATION=DEFAULT|NONE ))
```

Example

The fields generated from the following Linkage Section will not be decorated:

```
      >>ELK DECORATION=NONE
       Linkage Section.
       01 params.
          03 p1 pic x(10).
          03 p2 pic x(10).
```

HIDDEN Directive

This directive allows a field to be marked as hidden. This will avoid the need to convert a field into a service field.

```
>>ELK HIDDEN
```

or

```
$ELK HIDDEN
```

or

```
*(( ELK HIDDEN ))
```

or

```
*>(( ELK HIDDEN ))
```

Example

The field p1 will not be available in the web service:

```
       Linkage Section.
       01 params.
      >>ELK HIDDEN
          03 p1 pic x(10).
          03 p2 pic x(10).
```

INPUT Directive

The INPUT directive defines an input parameter for the web service. By default all the Linkage Section data items become input/output parameters. Use this directive to define an input only parameter. When the directive is placed on top of a group item, it's inherited by all the sub-items.

```
>>ELK INPUT
```

or

```
$ELK INPUT
```

or

```
*(( ELK INPUT ))
```

or

```
*>(( ELK INPUT ))
```

Example

In the resulting Web Service, the field p1 will be an input parameter, the field p2 will be an i/o parameter while the field p3 will be an output parameter.

```
       Linkage Section.
      >>ELK INPUT
       01 p1 pic x(10).
       01 p2 pic x(10).
      >>ELK OUTPUT
       01 p3 pic x(10).
```

MANDATORY Directive

The MANDATORY directive marks a field as a mandatory field. This information will be reflected in the WSDL file, therefore this directive has effect only for for Web Services of type SOAP.

```
>>ELK MANDATORY
```

or

```
$ELK MANDATORY
```

or

```
*(( ELK MANDATORY ))
```

or

```
*>(( ELK MANDATORY ))
```

Example

Specify that p1 is a mandatory field.

```
       Linkage Section.
       01 params.
      >>ELK MANDATORY
          03 p1 pic x(10).
```

NAME Directive

The NAME directive assigns a different name to the field defined on the next line. This directive can be used to avoid problems created by columns with incompatible or duplicate names. If the directive is used along with the INPUT or OUTPUT directives, it controls the name of the input or output parameter respectively.

Putting the name between quotes allows you to preserve hyphens. Without quotes, hyphens in the name are translated to underscore.

The name specified by this directive is used as is regardless of the decoration setting.

```
>>ELK NAME=fieldNname
```

or

```
$ELK NAME=fieldNname
```

or

```
*(( ELK NAME=fieldNname ))
```

or

```
*>(( ELK NAME=fieldNname ))
```

Example

Assuming that decoration is not disabled, the Web Service input will include the following parameters: param1, param2, p3_in. The Web Service output will include the following parameters: param1, p2_out, param3:

```
       Linkage Section.
       01 params.
      >>ELK NAME=param1
          03 p1 pic x(10).
      >>ELK INPUT, NAME=param2
      >>ELK OUTPUT
          03 p2 pic x(10).
      >>ELK INPUT
      >>ELK OUTPUT, NAME=param3
          03 p3 pic x(10).
```

The following snippet includes an equivalent syntax:

```
       Linkage Section.
       01 params.
      >>ELK NAME=param1
          03 p1 pic x(10).
      >>ELK INPUT
      >>ELK NAME=param2
      >>ELK OUTPUT
          03 p2 pic x(10).
      >>ELK INPUT
      >>ELK OUTPUT
      >>ELK NAME=param3
          03 p3 pic x(10).
```

NULLABLE Directive

The NULLABLE directive mars the field as nullable. Whenever the value of the field is empty, the field will be filled with the word 'null' before being returned in a JSON stream.

```
>>ELK NULLABLE
```

or

```
$ELK NULLABLE
```

or

```
*(( ELK NULLABLE ))
```

or

```
*>(( ELK NULLABLE ))
```

Example

In the resulting Web Service, the field p3 will be set to 'null' whenever the p3 data item is spaces:.

```
       Linkage Section.
      >>ELK INPUT
       01 p1 pic x(10).
       01 p2 pic x(10).
      >>ELK OUTPUT NULLABLE
       01 p3 pic x(10).
```

OPERATION Directive

The OPERATION directive specifies the name of the Web Service function. By default the name matches with the program name for programs without entry points and with the entry point name for programs with entry points.

```
>>ELK OPERATION=functionNname
```

or

```
$ELK OPERATION=functionNname
```

or

```
*(( ELK OPERATION=functionNname ))
```

or

```
*>(( ELK OPERATION=functionNname ))
```

Examples

The Web Service function will be named "vatcalculation" instead of "x0ag1".

```
       Identification Division.
       Program-Id. X0AG1.
       ...
      >>ELK OPERATION=vatcalculation
       Procedure Division.
```

The Web Service function will be named "vatcalculation" instead of "op01".

```
      >>ELK OPERATION=vatcalculation
       Entry "op01" using lnk-area01.
```

OUTPUT Directive

The OUTPUT directive defines an output parameter for the web service. By default all the Linkage Section data items become input/output parameters. Use this directive to define an output only parameter. When the directive is placed on top of a group item, it's inherited by all the sub-items

```
>>ELK OUTPUT
```

or

```
$ELK OUTPUT
```

or

```
*(( ELK OUTPUT ))
```

or

```
*>(( ELK OUTPUT ))
```

Example

In the resulting Web Service, the field p1 will be an input parameter, the field p2 will be an i/o parameter while the field p3 will be an output parameter.

```
       Linkage Section.
      >>ELK INPUT
       01 p1 pic x(10).
       01 p2 pic x(10).
      >>ELK OUTPUT
       01 p3 pic x(10).
```

TYPE Directive

The TYPE directive allows you to specify the data type of the service field. By default all parameters are strings. If the directive is used along with the INPUT or OUTPUT directives, it controls the type of the input or output parameter respectively.

```
>>ELK TYPE=type
```

or

```
$ELK TYPE=type
```

or

```
*(( ELK TYPE=type ))
```

or

```
*>(( ELK TYPE=type ))
```

Valid *type* values (case insensitive) are:

- base64Binary
- boolean
- byte
- decimal
- double
- float
- hexBinary
- int
- integer
- long
- short
- string
- unsignedByte
- unsignedInt
- unsignedLong
- unsignedShort

Example

Assuming that decoration is not disabled, the Web Service input will include the following parameters: p1_in(long), p2_in(long), p3_in(string). The Web Service output will include the following parameters: p1_out(long), p2_out(string), p3_out(long):

```
       Linkage Section.
       01 params.
      >>ELK TYPE=long
          03 p1 pic 9(5).
      >>ELK INPUT, TYPE=long
      >>ELK OUTPUT
          03 p2 pic 9(5).
      >>ELK INPUT
      >>ELK OUTPUT, TYPE=long
          03 p3 pic 9(5).
```

The following snippet includes an equivalent syntax:

```
       Linkage Section.
       01 params.
      >>ELK TYPE=long
          03 p1 pic 9(5).
      >>ELK INPUT
      >>ELK TYPE=long
      >>ELK OUTPUT
          03 p2 pic 9(5).
      >>ELK INPUT
      >>ELK OUTPUT
      >>ELK TYPE=long
          03 p3 pic 9(5).
```

USE GROUP Directive

The USE GROUP directive assigns a group of items to a single service field. The default data type for the resultant field is alphanumeric.

```
>>ELK USE GROUP
```

or

```
$ELK USE GROUP
```

or

```
*(( ELK USE GROUP ))
```

or

```
*>(( ELK USE GROUP ))
```

Example

Have a single field named "time" instead of three different fields named "hh", "mm", "ss".

```
       Linkage Section.
       01 params.
          03 p1 pic x(10).
      >>ELK USE GROUP
          03 time.
             05 hh pic 99.
             05 mm pic 99.
             05 ss pic 99.
```

VALUE Directive

The VALUE directive allows you to specify a constant value fo the field. If the directive is used along with the INPUT or OUTPUT directives, it controls the value of the input or output parameter respectively

The value is generated as is in the bridge program, so alphanumeric values should be enclosed between quotes, otherwise they will be treated as data items that you should define outside of the bridge program tagged areas.

```
>>ELK VALUE=fieldValue
```

or

```
$ELK VALUE=fieldValue
```

or

```
*(( ELK VALUE=fieldValue ))
```

or

```
*>(( ELK VALUE=fieldValue ))
```

Example

The field p1 will always have value 1. The field p2 will have value 1 in input. The field p3 will have value "ok" in output

```
       Linkage Section.
       01 params.
      >>ELK VALUE=1
          03 p1 pic 9(5).
      >>ELK INPUT, VALUE=1
      >>ELK OUTPUT
          03 p2 pic 9(5).
      >>ELK INPUT
      >>ELK OUTPUT, VALUE="ok"
          03 p3 pic x(2).
```

Chapter 12

SQL Directives

SQL directives are special comments that allow for the customization of the Embedded SQL blocks.

There are three syntaxes to specify an SQL directive.

Syntax 1:

```
>>SQL directive-name [=directive-value]
```

Syntax 2:

```
$SQL directive-name [=directive-value]
```

Syntax 3:

```
*(( SQL directive-name [=directive-value] ))
```

The >> marker can appeear everywhere except the sequence number area (columns 1 to 6) of the ANSI source format.

The $ and the * markers must be in the comment area, that is column 7 in ANSI source format and column 1 in Terminal source format.

If the source code is written in Free format, SQL directives can appear at any column with the following syntax.

Syntax 3:

```
*>(( SQL directive-name [=directive-value] ))
```

The directive must appear above the EXEC SQL statement that you want to customize.

Note that no spaces are allowed between the comment symbol and the couple of parenthesis. If spaces are present, the directive is treated as a standard comment and doesn't have effects.

SQL directives are ignored in Object and Factory methods. They can be used only in standard programs.

Splitting a SQL directive on multiple lines

A single SQL directive can be splitted on multiple lines by repeating the SQL delimiters on each line.

For example, the following snippets

Syntax 1:

```
>>SQL HOSTVAR
>>SQL 1,i;2,o
 exec sql
      execute
      begin
         stored_1 (:in-param, :out-param);
       end;
 end-exec
```

Syntax 2:

```
$SQL HOSTVAR
$SQL 1,i;2,o
 exec sql
      execute
      begin
         stored_1 (:in-param, :out-param);
       end;
 end-exec
```

Syntax 3:

```
*((SQL HOSTVAR))
*((SQL 1,i;2,o))
 exec sql
      execute
      begin
         stored_1 (:in-param, :out-param);
       end;
 end-exec
```

Syntax 4:

```
*>((SQL HOSTVAR))
*>((SQL 1,i;2,o))
 exec sql
      execute
      begin
         stored_1 (:in-param, :out-param);
       end;
 end-exec
```

are equivalent to

Syntax 1:

```
>>SQL HOSTVAR 1,i;2,o
 exec sql
      execute
      begin
         stored_1 (:in-param, :out-param);
       end;
 end-exec
```

Syntax 2:

```
$SQL HOSTVAR 1,i;2,o
 exec sql
      execute
      begin
         stored_1 (:in-param, :out-param);
       end;
 end-exec
```

Syntax 3:

```
*((SQL HOSTVAR 1,i;2,o))
 exec sql
      execute
      begin
         stored_1 (:in-param, :out-param);
       end;
 end-exec
```

Syntax 4:

```
*>((SQL HOSTVAR 1,i;2,o))
 exec sql
      execute
      begin
         stored_1 (:in-param, :out-param);
       end;
 end-exec
```

HOSTVAR Directive

The HOSTVAR directive specifies the type of the host variables in the Embedded SQL block. This directive is useful when a stored procedure is called in the Embedded SQL block and this stored procedure requires specific parameter types. It is particularly useful in the EXECUTE statement, where it's not possible to specify the parameter type after the parameter name like you can do in a CALL statement instead.

```
>>SQL HOSTVAR hostvarsTypes
```

or

```
$SQL HOSTVAR hostvarsTypes
```

or

```
*(( SQL HOSTVAR hostvarsTypes ))
```

or

```
*>(( SQL HOSTVAR hostvarsTypes ))
```

hostvarsTypes has the following format:

```
index,type[,dbtype[,typename]];index,type[,dbtype[,typename]];...;index,type[,dbtype[,
typename]]
```

Where

- *index* is the ordinal number of the host variable in the Embedded SQL block; use 1 to identify the first host variable, use 2 to identify the second host variable, and so on;
- *type* is the parameter type; use "i" for input, "o" for output and "u" for input-output.
- *dbtype* is the type of the corresponding database field. Any Java SQL type is allowed. Refer to the java.sqlTypes javadoc for the list of possible values. If the *dbtype* is ARRAY, specify also the type name; the Compiler will look for the configuration property iscobol.compiler.esql.array.TypeName in order to know which field type is used in the array.

It's not mandatory to describe all the host variables; those host variables that are not described are considered as *dbtype*=OTHER. The *type* instead is calculated as follows:

- host variables that are not used as parameter of a stored procedure are considered output if they're followed by ":=", input otherwise.
- host variables that are used as parameter of a stored procedure are treated according to the configuration property iscobol.compiler.esql.procedure.ProcedureName. If this property is not set, then they're considered as specified by iscobol.esql.default_param_type.

Example

Specify that in-param is an input parameter while out-param is an output parameter. Foo will be automatically considered as output since it's followed by ":=" :

```
      >>SQL HOSTVAR 2,i;3,o
       exec sql
           execute
           begin
              :foo := 1;
              stored_0 (:in-param, :out-param);
           end;
       end-exec
```

Chapter 13

Language Extensions

The isCOBOL Framework includes a serie of library routines, intrinsic functions and classes that provide additional features.

Please consult the following appendices for details:

- Library Routines
- Intrinsic Functions
- Internal Objects

Chapter 11

Object-oriented Programming

Some samples of Object-oriented programming are installed with isCOBOL. You can find them in the folder $ISCOBOL_HOME/sample/is-java.

Object oriented concepts

Object oriented programming consists of developing and implementing application systems as sets of interacting software objects.

A software object, as most objects in everyday life, such as an automobile, has a unique identity and certain attributes and behaviors. The automobile has a unique identity, its serial number; it has many attributes such as color, number of doors, and weight. It also has behaviors, such as forward, reverse, accelerate, shift, and the like. Software objects are used to model real world objects and as such they abstract the key concepts of the real world object in software. A software object used to model an automobile would for example, have a unique identity, and attributes such as color, weight, and length, as well as behaviors such as forward and reverse.

Software objects can be used to model any of the concepts germane to a given problem domain. For example, they can represent bank accounts, employees, parts, processes, programs, fields, files, structures and the like.

Therefore, we can say a software object is an entity that has a unique identity, specific data values, and specific behaviors or program code. The program code is organized into small modules. In object oriented terminology these modules are called methods. Data is encapsulated within each object and can only be accessed by using one or more of the object's methods.

Classes

To facilitate dealing with the hundreds or even thousands of different software objects that can exist in an application system, objects are organized into classes. A class is a group of objects that have a common data structure and that all use the same methods. This means that the data structure can be defined for the class. Each object within the class has a unique set of data values that correspond to the class structure. It also means that the methods are defined once at the class level and are used by each of the objects of the class.

> Example:
>
> A banking application will require many individual accounts. Each account will have data associated with it, for example account-balance and date-opened. Each account will have methods that allow other parts of the application to access an individual account such as deposit, withdraw and current-balance. The checking account class defines the data layout and methods for all of the individual checking accounts, thus they all work in the same way and serve the same purpose.

Objects

Every object belongs to exactly one class; there may be zero, one, or more objects in any given class.

Object instantiations

The individual members of a class are called instance objects, or simply instances. For example, an individual checking account object can be called a checking account object instance, or a checking account instance. We will use the term instance to refer to an individual member of a class.

All instances of a class share the same data definition, but each instance has its own unique values.

The instances of a class also share the same methods. A method is defined once for the class, but each instance behaves as if it is the sole owner of the methods defined for the class. The methods are shared and can be used by all instances. This facility of object oriented programming environments permits the data for each instance to be hidden (encapsulated) because it can be accessed only via the methods of the class.

The conventions used in all of the code fragments and the sample bank application are as follows:

- Class names and interface names use camel case, the first letter of each word is capitalized. For example, the class checking account is "CheckingAccount".
- Method names use lower case for the first word and camel case for subsequent words, for example, the method deposit is "deposit"; and the method calculate charges is "calculateCharges".
- Data item names always consist of at least two lower case words separated by a "-", for example, customer name is "customer-name", an object is "an-object", and so forth.

Example:

Each checking account is represented by an account instance. Each instance has its own copy of the data described by the class, the customer's name, the current balance, and the date opened. Each instance uses the methods defined for the class to carry out its functions, for example, the deposit method credits the account.

Within the body of program code that defines a class, instances of the checking account class could be defined as follows:

```
...
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01   checking-account.
     03 customer-name    PIC X(35).
     03 current-balance  PIC S9(9)V99.
     03 date-opened      PIC 9(8).
...
PROCEDURE DIVISION.
METHOD-ID. deposit.
     method code
END METHOD.
...
METHOD-ID. withdraw.
     method code
END METHOD.
...
END OBJECT
```

Object data definitions

Since each instance object is unique, it has a unique reference value that is generated by the runtime system when the instance is created. An object's reference value serves as a pointer to that specific instance. An object reference value can be thought of as a key that identifies a specific instance. The usage clause provides a means to define a data item, called an object reference, to hold an object's reference value. In the following example, data items a-checking-account and an-account are object references:

Example:

```
...
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  checking-account.
    03 customer-name    PIC X(35).
    03 current-balance  PIC S9(9)V99.
    03 date-opened      PIC 9(8).
01  a-checking-account  USAGE IS OBJECT REFERENCE.
01  an-account          USAGE IS OBJECT REFERENCE.
PROCEDURE DIVISION.
...
```

The data item a-checking-account can be used to refer to a specific instance. A data item that has been defined as an object reference can be set equal to another data item defined as an object reference through the use of the SET statement as follows:

```
SET an-account TO a-checking-account.
```

The above statement will transfer the value of the object reference in a-checking-account to an-account.

Object references

An object reference is a data item that contains a reference to an object. The content of the object reference is used to locate the object and information associated with the object.

An application may ensure at compile time that an object of one class, say employee, will never be used as an object of an unrelated class, say account. This is done by using an object reference described with either a classname, an interface-name, or an ACTIVE-CLASS phrase. If a class-name is specified, the data item can only be used to reference an object of the class specified, or one of its subclasses, as discussed in Inheritance. If an interface-name is specified, the data item can only be used to reference an object described with an IMPLEMENTS clause that references the interface specified, as discussed in Class polymorphism. If the ACTIVE-CLASS phrase is specified, the data item can only be used to reference an object of the same class as that of the object with which the method was invoked. Object references for ACTIVE-CLASS are of special significance as returning items. This capability is needed for defining a new method in the BASE class that works as documented without violating the conformance rules, and it allows writing user methods that do object creation in conformance with the definition of the class hierarchy.

Alternatively, an application may use a universal object reference, one that can refer to any object. A data item is defined as a universal object reference by omitting all optional phrases from the OBJECT REFERENCE phrase of the USAGE clause. Runtime validation may be used to ensure that an object has the correct interface as described in Conformance for parameters and returning items, but this approach does require more runtime resources.

Examples:

The definition of a data item that can refer only to an object of the CheckingAccount class or one of its subclasses is:

```
01 an-account USAGE IS OBJECT REFERENCE CheckingAccount.
```

The definition of a data item that can hold a reference to any object is as follows:

```
01 an-object USAGE IS OBJECT REFERENCE.
```

Factory objects

As stated previously, a class describes the data for each instance of the class and defines the methods that can be used by each instance of the class. Each class has one object, called the factory object, that is responsible for functions, such as creating a new instance of the class and managing data associated with all instances of the class.

A factory object can be thought of as an instance of a special kind of class and has data (factory data) and methods (factory methods). The data and methods for the factory object are defined as part of the class definition.

Every instance of a class is created by the factory object of that class. When an object is created, the data descriptions in the class are used to allocate storage for the instance.

Example:

The checking account class shown below describes a factory object, which is called the checking account factory object. To create a new checking account instance, a method in the checking account factory object is used. To keep track of the number of checking account instances a data item in the factory object can be used. Whenever a new instance is created, 1 can be added to the value; whenever an instance is removed, 1 can be subtracted from the value.

Sample code for the checking account factory object could be as follows:

```
CLASS-ID. CheckingAccount INHERITS Base.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
   CLASS Base.
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  number-of-accounts          PIC 9(5).
PROCEDURE DIVISION.
METHOD-ID. newAccount.
create-account.
   add 1 to number-of-accounts.
...
END METHOD.
END FACTORY
OBJECT.
...
```

Methods

A method is procedural code that defines a specific function required by all of the instances of a class. A method may be thought of as a module or subroutine. A class may define as many methods as it needs to manage the data defined for the class. Methods are typically only a few lines of procedural code, but may be as many lines as required to accomplish a specific function.

An instance object is used by invoking one of its methods. This facility is similar to the call facility. With conventional coding techniques, one program activates another program by issuing a call. With object oriented programming techniques, one object activates another object by issuing an invoke.

Methods are distinguished by their name alone. A class cannot define two methods with the same name. If a subclass defines a method with the same name as a method in an inherited class, the method in the inheriting class overrides the inherited method.

Method invocation

Any program or method can invoke a method to act on an object. The name of the method specified on the invocation statement will be the method executed. The invocation statement also allows arguments to be

passed to the method and also allows the method to return a result.

Example:

Whenever an application needs to use an object, it invokes a method to act on the instance object. Let's assume the CheckingAccount class contains the methods deposit, withdraw and balance and that an-account references an instance of the Account class. The syntax to deposit an amount to an account is as follows:

```
    INVOKE an-account "deposit" USING in-amount
```

```
Similarly, the syntax to determine the current balance of an account is:

    INVOKE an-account "balance" RETURNING current-balance
```

```
An equivalent statement illustrating inline method invocation is:

    MOVE an-account:>balance TO current-balance
```

When the application needs to determine the balance of a specific account, a conventional program or a method will request the instance to activate its balance method. Code fragments to accomplish this are shown below:

Assume a program wants to determine the balance of a checking account.

Program Code

```
    WORKING-STORAGE.
    ...
    01 a-checking-account-object USAGE IS OBJECT REFERENCE CheckingAccount
    ...
    77 the-balance PIC S9(8)V99 VALUE ZERO.
    ...
    PROCEDURE DIVISION.
    ...
       INVOKE a-checking-account-object "balance" RETURNING the-balance. *> assume the object
                                                  *> referenced by a-checking-account-object
                                                  *> contains the reference to the desired
                                                  *> account
```

Checking Account Class

```
    ...
    OBJECT.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
```

```
01  checking-account.
    03 customer-name    PIC X(35).
    03 current-balance  PIC S9(9)V99.
    03 date-opened      PIC 9(8).
...
PROCEDURE DIVISION.
...
METHOD-ID. balance.
DATA DIVISION.
...
LINKAGE SECTION.
01  ls-balance          PIC S9(8)V99.
...
PROCEDURE DIVISION RETURNING ls-balance.
return-balance.
   MOVE current-balance TO ls-balance.
   EXIT PROGRAM.
END METHOD.
...
```

Method prototypes

Method prototypes are method skeletons that define the method name, parameters, and information necessary to describe those parameters such as that specified in the `REPOSITORY` paragraph and the `SPECIAL-NAMES` paragraph. They do not include the procedural code. In essence they provide all of the information required to invoke a method. Method prototypes are used to specify interfaces and are specified in interface definitions.

Example:

The method prototype for the calculateInterest method of the SavingsAccount class is as follows:

```
METHOD-ID. calculateInterest.
DATA DIVISION.
LINKAGE SECTION.
01  interest-rate       PIC S9(3)v9999.
01  interest-amount     PIC S9(9)V99.
PROCEDURE DIVISION USING interest-rate RETURNING interest-amount.
END METHOD.
```

Unnamed methods

Unnamed methods are methods without name. An unnamed method is declared by giving him an empty physical name. The following syntaxes are equivalent and produce unnamed methods:

1)

```
METHOD-ID. somename as "".
```

2)

```
METHOD-ID. "".
```

An unnamed method cannot have arguments, it cannot return values and it cannot raise exceptions. The keywords OVERRIDE, PUBLIC, PRIVATE and PROTECTED are ignored by the Compiler in this context.

These methods can be both factory methods or instance methods: in the former case the method is invoked when the class is created before the invocation of any other class method; this feature is also known as "static initialization block" or "static initializer". In the latter case the method is invoked before any constructor; this feature is also known as "non-static initialization block" or "non-static initializer".

Unnamed methods are useful to define constants. For example, the following code:

```
       IDENTIFICATION DIVISION.
       METHOD-ID.  "".
       WORKING-STORAGE SECTION.
       LINKAGE SECTION.
       PROCEDURE DIVISION.
       MAIN.
           set const1 = 13.
       END METHOD.
       END FACTORY .
```

produces the following constant in the class:

```
public static int CONST1;
```

Other object oriented programming features

Some of the other capabilities are Inheritance, Interfaces and Polymorphism, and Conformance.

Inheritance

One of the language features that separates object oriented languages from conventional programming languages is the ability to develop a hierarchy of classes, as shown by the example below.

The manager class is a subclass of the employee class which in turn is a subclass of the person class. Or said another way, the employee class is the superclass of the manager class and the person class is the superclass of the employee class. At any point in the hierarchy, the classes above a given class are its superclasses or its ancestors and the classes below are its subclasses or its children. A subclass includes all of the capabilities of all of its ancestors and additionally may add to or override these capabilities. For example, the methods of the person and employee classes are available to an instance of the manager class as well as the methods defined for the manager class.

Inheritance is the mechanism used to develop class hierarchies.

Inheritance supports a hierarchy of classes, where every instance of a subclass can be used "as if it were" an instance of its superclasses. For example, a manager object could be used as if it were an employee object, and an employee object could be used as if it were a person object. This is the principle of conformance between classes. When classes conform, a data item declared as a reference to an object of a given class may in fact reference an object of any class that descends from that given class.

Inheritance represents an "is a" relationship between two classes and is a way of specializing a higher level class. In the figure above, Manager class, a manager "is an" employee and an employee "is a" person. All the object data definitions described in the superclasses, person and employee, plus the object data definitions for the class itself, manager, are used to create an instance of the manager class. Also, the methods defined by the superclasses are inherited by the subclass and are used to directly operate on any instance of manager.

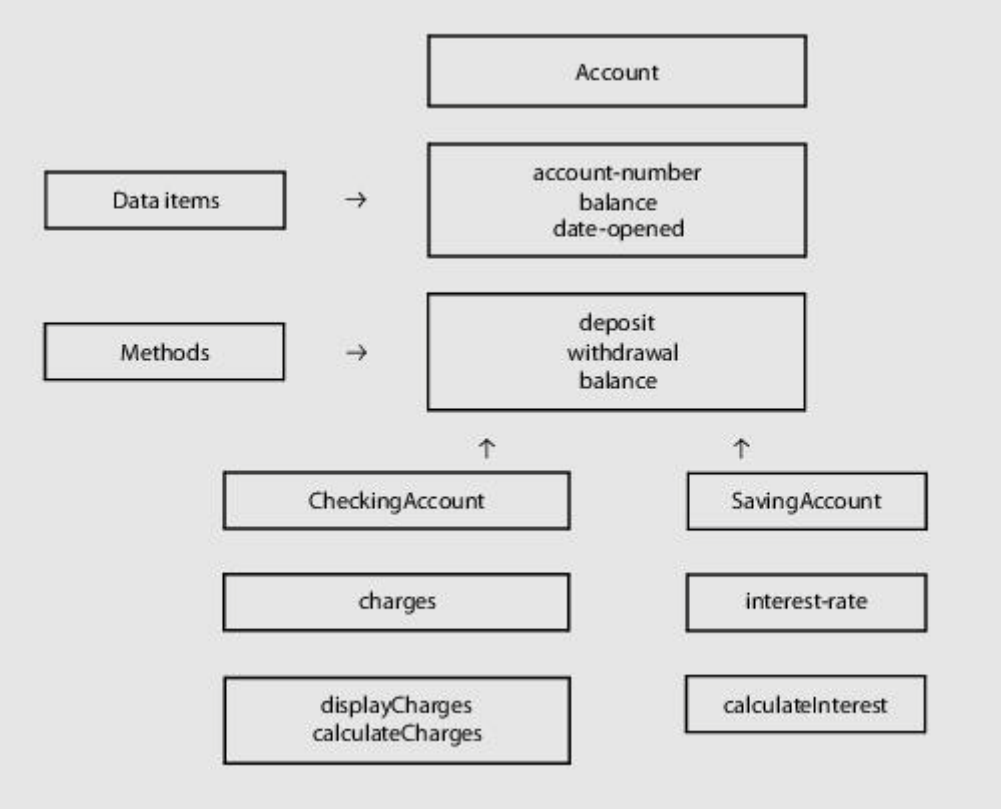
Both factory and instance data and methods of all ancestor classes are inherited by a class that inherits from another class.

When inheritance is used to define a subclass, data in the superclass is encapsulated because methods defined for the subclass are not allowed to directly access the data items defined for the superclasses. It requires all subclasses to use methods defined for the superclasses to access the data items defined for the superclasses. As an example, if the employee class defined the data item "employee-name", the employee class would have to include a method, say getName, to allow any subclass to retrieve the employee name.

A class may inherit from more than one other class, this is called multiple inheritance.

Example of single inheritance:
A bank will have different kinds of accounts, and yet they are all accounts. If we consider checking accounts and savings accounts, they have a number of common features, they both have an owner and a balance. They also have some different features, the fact that checks are allowed for one and the other pays interest. It makes sense to have a basic account class that contains the common parts and then to use inheritance to define the checking account and the savings account subclasses. Thus the account class defines what is common to all accounts; the checking account class defines what is specific to checking accounts; and the savings account class defines only what is specific to savings accounts. Any changes to the account class will be picked up by the inheriting classes automatically.

These relationships can be represented as shown in the figure below.



In the example shown, each instance of CheckingAccount is automatically created with memory allocated for the attributes account-number, balance, date-opened and charges. Additionally, the methods deposit, withdraw, and balance inherited from Account and the methods displayCharges and calculateCharges defined in the CheckingAccount class can act on each instance. Each instance of SavingsAccount is automatically created with memory allocated for the attributes account-number, balance, date-opened and interest-rate. Each instance of SavingsAccount can access the methods deposit, withdraw and balance inherited from Account and the method calculateInterest defined for itself.

Some sample code for the account and checking account classes is shown below:

Account Class

```
CLASS-ID. Account INHERITS Base.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY
   CLASS Base.
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  number-of-accounts      PIC 9(5).
PROCEDURE DIVISION.
METHOD-ID. newAccount.
   method code
END METHOD.
```

```
...
END FACTORY.
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  ACCOUNT-INFORMATION.
    03 account-number       PIC X(12).
    03 balance              PIC S9(8)V99.
    03 date-opened          PIC 9(8).
...
PROCEDURE DIVISION.
METHOD-ID. deposit.
...
END METHOD.
METHOD-ID. withdraw.
...
END METHOD.
METHOD-ID. balance.
...
END METHOD.
...
```

CheckingAccount Class

```
CLASS-ID. CheckingAccount INHERITS Account.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
CLASS Account.
...
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  checking-account
    03 charges              PIC S9(8)V99.
...
PROCEDURE DIVISION.
METHOD-ID. displayCharges.
...
END METHOD.
METHOD-ID. calculateCharges.
...
END METHOD.
...
```

Restricting inheritance and modification with the FINAL clause

It may be desired that no extension be done to a class through inheritance. This, for example, might be needed by library providers who want to control the implementation of a class. To provide this functionality, a class may be declared 'final' if its definition is complete and no subclasses are desired or required. A compile-time error occurs if the name of a final class appears in the `INHERITS` clause of another class declaration; this implies that a 'final' class cannot have any subclasses.

Because a final class never has any subclasses, the methods of a final class are never overridden. This may be an overkill in some cases, and it might be desired that only a few methods not be overridden. For that purpose, we can use the attribute 'final' with a method of any class, to prohibit the subclasses of that class from overriding that method. This attribute can also be used redundantly with the methods of a final class.

Restricting methods from being overridden also helps in the 'pairing' of methods of a class. Here's an example: Suppose a class A defines a method 'bar' calling another method 'foo' defined in the same class. If there is a subclass B that also defines 'foo', and if we invoke 'bar' on an object of class B, the method 'bar' (inherited from class A) will end up calling the function 'foo' defined for B and not the original 'foo'. If however, we specify the `FINAL` clause with the function 'foo' in class A, it cannot be overridden in class B, and the programmer can ensure that 'bar' will always invoke the same 'foo'.

The `FINAL` attribute has to be handled carefully while dealing with multiple inheritance. If two classes A and B define a method of the same name, and if a class C inherits from both of them, the method definitions are not allowed to have the `FINAL` clause specified. However, if the same method is inherited through two classes which had the same superclass defining that method, the method is allowed to have the `FINAL` clause specified in its definition. This would happen for a diamond shaped multiple inheritance, where classes B and C inherit from a class A, and then class D inherits from both B and C. Class A can have methods with the `FINAL` clause specified, and though D will appear to inherit two methods of the same name with the `FINAL` clause, it is acceptable as they are the same method implementations.

Conformance

Conformance allows the compiler to check the application code and determine if any class hierarchy inconsistencies can occur at runtime. A data item can be constrained to be an object reference for only objects of a specific class or its subclasses by inserting the class name after the `USAGE IS OBJECT REFERENCE` clause. Additionally, the compiler can determine if the arguments passed to a method match the parameters specified in the `USING` phrase of the procedure division header. Arguments must match the parameters exactly to ensure conformance.

Example:

One part of a banking application can be written to deal with any kind of account. Data items that have been declared such as:

```
01 an-account USAGE OBJECT REFERENCE Account.
```

can reference any object of the account class or any object of a class that inherits from the account class. The rules of conformance ensure that the subclasses of the account class can be used in exactly the same way as objects of the account class itself. The underlying implementation of a subclass may be different from the original account class, but the interface is guaranteed to be compatible.

A different part of the banking application can be written to deal only with a specific kind of account. For example, the data item an-account would be defined as follows to hold a reference to a checking account object or to any object of a class that inherits from the CheckingAccount class. Note that in this example, no classes inherit from the CheckingAccount class.

```
01 an-account USAGE OBJECT REFERENCE CheckingAccount.
```

If a source element contains code that attempts to put a reference to an Account object into the data item declared to contain a checking account object, the compiler will warn the user that there is a potential error. It should be noted that the compiler cannot necessarily determine that this actually is an error, only that it might be an error. Additionally, the compiler can check to ensure that the arguments and parameters match.

The policy for conformance checking is conservative and errs on the side of caution.

These are some examples of restrictions imposed by compile time conformance checking, even though at runtime a conformance violation might not actually exist:

1. Let's assume there is a class A with a subclass A1, and a source element containing the following definitions:

```
01 or-1 object reference A.
01 or-2 object reference A1.
```

   The statement

```
SET or-2 to or-1
```

   is invalid, because or-1 may contain, for example, a reference to class A, which is not valid in or-2. At runtime, or-1 might actually contain a reference to A1, which would be a valid content of or-2. This is, however, not predictable at compile time. Therefore, the `SET` rules require that the class of the sending operand, A in this case, is the same class or a subclass of the class of the receiving operand (A1), which is not the case.

2. Although returning an object reference for ACTIVE-CLASS is generally allowed, there are still restrictions due to the compile-time checking requirement. The compiler does not know the object that will be used to invoke the method; it does, however, know the object reference that is used to invoke the method containing the object reference to be returned. It is possible for the compiler to derive some information about the item to be returned.

   Consider the following class:

```
Class-id. C inherits B.
Factory.
   Method-id. M
   Working-Storage section.
   01 or-1 object reference active-class.
   Procedure division returning or-1.
   End method.
End class.
```

   Consider:

```
Invoke C "M" returning anObj.
```

   The compiler knows that the object returned by method M is of class C or some subclass of C. This knowledge can be used to detect some errors. For example, suppose we have this class hierarchy:

```
B
|
C
|
D
```

Consider the following statements:

```
01 or-B object reference B.
01 or-C object reference C.
01 or-D object reference D.

Invoke C "M" returning or-B
Invoke C "M" returning or-C
Invoke C "M" returning or-D
```

The compiler can statically determine that the first and second invokes are valid but the 3rd invoke is invalid, since it might result in storing an object of type C in an object reference of type D.

3. In principle, method invocation on a specific object identified at runtime is permitted for any method that is defined for that object. Compile time checking, however, restricts the eligible methods to those that are known at compile time. For example, if the specified object reference is restricted to a specific class, a method with this name must be defined in the class specified in the object reference. Thus it is possible to invoke an overriding method defined in a subclass of the specified class, but not a method that is defined only in the subclass, but not in the parent class.

   Let C-1 be a class with a subclass C-2, where C-1 contains a method M-1 and C-2 contains a method M-2. Assume further a client program or method contains:

```
01 or-1 object reference C-1.

Invoke or-1 "M-1"
```

   This is valid. Even when or-1 actually references an object of C-2, there is no problem, because it is still the same M-1 in class C-1. Also, there is no problem when M-1 is defined in C-2 as a method overriding the M-1 of C-1, because the signature of the overriding M-1 is still the same.

   But:

```
Invoke or-1 "M-2"
```

   is invalid, even when or-1 actually references an object of C-2, because the signature of M-2 is not known at compile time, and there is no way of conformance checking at compile time.

   Note, however, that the object modifier can be used to get type-safe access to M2, with conformance checking at runtime:

```
Invoke or-1 as C2 "M-2"
```

Polymorphism

Polymorphism is supported by COBOL in two different ways. Class polymorphism is supported through class inheritance and the use of interfaces. Parametric polymorphism is supported through method overloading.

Class polymorphism

Class polymorphism is generally provided through the class inheritance. In COBOL, the use of interfaces also provides class polymorphism, too.

An interface definition defines a subset of methods of any class implementing that interface. It provides a view of the methods that can be invoked for the class, including the names and parameter specifications for each method. That is, only method prototypes are described in the source unit of an interface definition.

A class may implement several interfaces. Each interface may include one or more of the methods of that class.

An object that implements all of the methods defined in an interface conforms to that interface. The application class hierarchy forms a hierarchy of conforming interfaces.

Example:

A banking application may have defined a method in the Account class that prints the data values associated with each instance, for example, current owner and balance. Likewise, a method in the Customer class can print the name and address of the customers it represents. If there is a need for a generalized routine that prints things, with correct page formatting, a class can be defined that contains the methods associated with printing. Any object that implements this class can then be printed by this routine. This illustrates polymorphism.

Sample code for the Print Class is shown below:

```
CLASS-ID. PrintReport.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
PROCEDURE DIVISION.
METHOD-ID. printRpt.
END METHOD.
...
```

Ad-Hoc polymorphism

Through the use of the `REDEFINES` clause a programmer can pass different classes of data to a method at runtime. As long as the data passed to the method is the same length, the determination of what data is passed can be determined through class condition tests in the method.

Example:

```
Method-id. AddIt.
Working-Storage section.
01  Out.
    02 Out-X                  pic X(4).
    02 Out-9 redefines Out-X  pic 9(4).
Linkage section.
01  In1.
    02 In1-X                  pic XX.
    02 In1-9 redefines In1-X  pic 99.
01  In2.
    02 In2-X                  pic XX.
    02 In2-9 redefines In2-X  pic 99.
Procedure division using In1, In2 returning Out.
    If In1-9 is numeric and In2-9 is numeric
        Compute Out-9 = In1-9 + In2-9
    Else
        String In1-X, In2-X delimited by size into Out-X
    End-if.
    End method.
```

Invoking AddIt with the input parameters of 'AB' and 'CD' would produce the result 'ABCD', whereas invoking AddIt with the input parameters of '01' and '02' would produce '0003'.

Unfortunately, object references cannot be redefined. This limits COBOL's support of ad-hoc polymorphism to items of a class and category that can exist within a weakly typed structure.

Parametric Polymorphism

Method overloading provides the ability to declare two or more methods of the same name, but with a different number of parameters and types. Each method has a method resolution signature that consists of the method name and all of the relevant information from the definition of each of the parameters, and from the returning item. During the resolution of the method, the signature derived from the invocation is compared with all methods of the same name. If the signatures match exactly, that method is bound. If not, the method that most closely matches the signature while still conforming is bound.

No two methods within a class may have the same signature.

Example:

Suppose we have three methods that print. The first method invokes the print method of whatever object pointer is passed to it. The second formats a packed decimal field and prints it, the third prints a variable length character string.

```
Method-id. PrintIt.
Linkage section.
01  In-o                   usage object.
Procedure division using In-o.
    invoke in-o "PrintMe".
End Method.
Method-id. PrintIt.
Working-Storage section.
01  Out-p                  pic ZZZ,ZZZ,ZZZ.
Linkage section.
01  In-p                   pic 9(9).
Procedure division using In-p.
    move In-p to Out-p.
    display out-p.
End Method.
Method-id. PrintIt.
Working-Storage section.
01  Out-p                  pic $$$,$$$.99.
Linkage section.
01  In-p                   pic s9(5)v99 packed-decimal.
Procedure division using In-p.
    move In-p to Out-p.
    display out-p.
End Method.
Method-id. PrintIt.
Linkage section.
01  In-x                   pic x(20).
01  In-len                 pic 9(4).
Procedure division using In-x, In-len.
    display in-x(1:in-len).
End Method.
```

If we invoke PrintIt with an object reference as in

```
    Invoke aClass "PrintIt" using anObject.

or
    aClass:>PrintIt (anObject).
```

we would invoke the first method, which would invoke the "PrintMe" method of anObject. If we invoked PrintIt as follows:

```
    Invoke aClass "PrintIt" using "FooBar", 3.

or
    aClass:>PrintIt ("FooBar", 3).
```

we would invoke the last method, and display "Foo".

Invoking Printit with a numeric as follows:

```
    Invoke aClass "PrintIt" using 32

or
    aClass:>PrintIt (32)
```

could match either the second or the third method, since the literal "32" is treated as USAGE DISPLAY. When there are two methods that match equally well, the first of the methods that match is the method to which we will resolve. In this case, it would be the second method, and we would display "32". Placing the method that is preferred first in the compilation unit assures that the method will be select when the choice is ambiguous.

Boxing

Boxing, otherwise known as wrapping, is the process of placing a primitive type within an object so that the primitive can be used as a reference object.

Autoboxing

Autoboxing is the term for getting a reference type out of a value type just through type conversion (either implicit or explicit). The compiler automatically supplies the extra source code that creates the object.

For example, without autoboxing, the following code would not compile:

```
       configuration section.
       repository.
           class jShort as "java.lang.Short".
       working-storage section.
       77 s object reference jShort.
       procedure division.
       main.
      *    set s = jShort:>new(5). |always ok
           set s = 5.
```

Without autoboxing, the Compiler would not accept the last line. Short are reference objects. To convert from a primitive integer value to a Short, you should "manually" instantiate the Short object. With autoboxing, instead, the Compiler accepts the last line, and automatically transforms it so that a Short object is created to store the value 5.

Unboxing

Unboxing refers to getting the value that is associated to a given object, just through type conversion (either implicit or explicit). The Compiler automatically supplies the extra source code that retrieves the value out of that object, either by invoking some method on that object, or by other means.

For example, without unboxing, the following would not compile:

```
       configuration section.
       repository.
          class jShort as "java.lang.Short".
       working-storage section.
       77 s object reference jShort.
       procedure division.
       main.
      *    if s:>shortValue() = 5 |always ok
           if s = 5
              continue
           end-if.
```

Without unboxing, the Compiler would not accept a comparison between a Short object and a primitive integer number. With unboxing, instead, the Compiler accepts the comparison and automatically transforms it so that a numeric value is extracted from the Short object.

Object management

Objects

As mentioned previously, objects are allocated at runtime and are accessed only through object references. An object is created by invoking a method in a factory object. The object continues to exist until it can no longer be accessed. The object can no longer be accessed when no object reference data item references that object.

Class library

A small class library is provided that can be used, extended, or integrated into business solutions. The `BASE` class is the single top node of the class library and includes methods that support object creation and initialization, and also other primitive functions useful for objects.

Parameterized classes

A parameterized class is a skeleton class definition that when passed the appropriate parameters will create a new class tailored to perform a given function. Parameterized classes allow developers to create classes that have common behavior. For example, a single parameterized collection class can be used to define many types of collection classes.

Example:

Let's consider container classes which are typically used to hold references to objects of a given class. In a banking application, a specific type of container class, say AccountCollection, could be used to hold the list of identifiers to each account object. To generate a collection class that is capable of holding or managing only identifiers of checking account objects, the class name CheckingAccount is specified as a parameter when the collection class is created.

Suppose AccountCollection is a parameterized class whose definition is given elsewhere. Part of its class definition is as follows:

```
CLASS-ID. AccountCollection INHERITS Base USING X.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
CLASS Base
CLASS X.
....
```

In another source unit, the following would create a class which is a collection of checking accounts and an object reference for this new kind of class.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
CLASS CheckingAccountCollection EXPANDS AccountCollection USING CheckingAccount.
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  a-checking-account-collection USAGE OBJECT REFERENCE CheckingAccountCollection.
...
```

Files in object orientation

Files in object oriented applications can be specified in two different ways:

1. Files in instance objects
2. Files in factory objects

Files in instance objects

A file specified in an instance object means that the instance definition contains the `FILE-CONTROL` paragraph and the file section. One or more instance methods will contain the file processing statements such as OPEN, CLOSE, READ and WRITE. All of the instance methods have visibility to the records associated with the file.

When a class that contains a file specified in the instance definition is inherited, each direct or indirect descendent also inherits the file specification. The instance objects of each of these subclasses have their own file connector unless the EXTERNAL clause is specified for the file. Dynamic file assignment or file sharing may be used to resolve conflicts in accessing the physical files associated with these file connectors.

Specifying dynamic file assignment in the file control entry permits a class to be used to define a logical file of a given structure, and each instance can associate a different physical file with its own file connector and perform I-O on that physical file.

Sample code for dynamic file assignment is illustrated below. Note that the MOVE statements only have an effect on the dynamic assignment when a subsequent OPEN statement for the file connector is executed.

```
CLASS-ID. Employee INHERITS Base.
...
OBJECT.
...
FILE-CONTROL.
    SELECT EMPLOYEE-FILE ASSIGN USING FILE-REF.
DATA DIVISION.
FILE SECTION.
FD  EMPLOYEE-FILE
...
WORKING-STORAGE SECTION.
01  FILE-REF               PIC X(16) VALUE SPACES.
...
PROCEDURE DIVISION.
METHOD-ID. readFile.
...
WORKING-STORAGE SECTION.
01  EMPLRCD.
    03 SSN                 PIC 9(9).
    03 NAME ...
PROCEDURE DIVISION.
...
    MOVE "external-ref01" TO FILE-REF
    OPEN INPUT EMPLOYEE-FILE
...
    READ EMPLOYEE-FILE NEXT RECORD INTO EMPLRCD
    CLOSE EMPLOYEE-FILE
    ...

    MOVE "external-ref02" TO FILE-REF
    OPEN INPUT EMPLOYEE-FILE
    ...
    READ EMPLOYEE-FILE NEXT RECORD INTO EMPLRCD
    ...
```

Files in factory objects

When a file is specified in a factory, this means that the factory definition contains the FILE-CONTROL paragraph and FILE SECTION. One or more of the factory methods will contain the file processing statements such as OPEN, CLOSE, READ and WRITE. All of the factory methods have visibility to the data on the file without the use of the EXTERNAL clause.

Inherited Factory Object Definitions

It is important to note that when a class that contains a file specified in the factory object definition is inherited, each direct or indirect descendent also inherits the file specification. As in the case of inherited object definition above, the factory objects of each of these subclasses have their own file connector unless the `EXTERNAL` clause is specified for the file. Dynamic file assignment or file sharing may be used to resolve conflicts in accessing the physical files associated with these file connectors.

Data items in object orientation

1. Data items defined in Instance Objects are local for the single instance.
2. Data items defined in Factory objects are shared between all the programs in the runtime session (in isCOBOL Server environment they're shared between all clients).

Exception objects

Exception objects are used similarly to predefined exception conditions. They provide, however, more flexibility and allow access to a wider variety of information for resolving exception situations, compared to the predefined or user-defined exception conditions, especially when used in object oriented applications.

Although in principle any object of any class could serve as an exception object, there will more likely be a specific class for a specific kind of exception. This allows the application to invoke a method tailored for handling that exception.

There might be, for example, a class called `INVALID-ACCOUNT`, whose objects correspond to individual occurrences of the invalid account condition. The application can create and "raise" such an object when the application detects a transaction with an invalid account number. Let's assume the object reference pointing to this object is called `AN-INVALID-ACCOUNT`. As soon as the application method (assuming it is a method rather than a program) detects the error, it can execute a `RAISE AN-INVALID-ACCOUNT` statement.

Note that the `RAISE AN-INVALID-ACCOUNT` returns the exception to the invoking runtime element only if this use of the class `INVALID-ACCOUNT` has been "announced" by listing that class (or a superclass of it) in the procedure division header of the method containing the RAISE statement.

Similarly, an interface-name rather than a class-name may be specified in the procedure division header of a source element containing a RAISE statement. In this case, the object being raised shall be described with an `IMPLEMENTS` clause that references the specified interface in order to qualify for propagation.

Lambda expressions

A lambda expression (also known as anonymous function) is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

The syntax of a lambda expression is:

```
->expression
```

isCOBOL supports lambda expressions inside COBOL classes by allowing you to specify a method name instead of a data-item name under the following conditions:

1. the type of the parameter must be a functional interface, that is an interface with only one method,
2. the method referenced via lambda must have the same signature (parameters, return type and thrown exceptions) of the method in the functional interface,
3. there must not be any space between '->' and the method name next to it,

4. the method must be referenced without parameters,
5. the method name must be univocal in its class.

Note - Lambda expressions are not supported in standard COBOL programs with PROGRAM-ID in their IDENTIFICATION DIVISION. They can be used only in COBOL classes with CLASS-ID in their IDENTIFICATION DIVISION.

Example with Java functional interface

Java has several functional interfaces built in. One of these is FilenameFilter. Instances of classes that implement this interface are used to filter filenames. A FilenameFilter class is required for example by the list method of java.io.File.

The small COBOL class below lists all the cbl files in the current directory by printing their name on the system output. Files that don't have a cbl extension are not listed. The filtering is performed by the filterFileName()

method, that matches the accept method in the FilenameFilter interface and is invoked via lambda:

```
identification division.
class-id. myclass as "myclass".

environment division.
configuration section.
repository.
   class j-string     as "java.lang.String"
   class j-string-arr as "java.lang.String[]"
   class j-io-file    as "java.io.File"
   class j-system     as "java.lang.System"
   .
identification division.
factory.

procedure division.

identification division.
method-id. main as "main".
working-storage section.
77 curr-dir  object reference j-io-file.
77 file-list object reference j-string-arr.
77 len       int.
77 i         int.
linkage section.
01 args      object reference j-string-arr.
procedure division using args.
main.
   set curr-dir to j-io-file:>new(".").
   set file-list to curr-dir:>list(->myclass:>filterFileName).
   set len to file-list:>length.
   perform varying i from 0 by 1 until i >= len
      display file-list(i)
   end-perform
   goback.
end method.

identification division.
method-id. filterFileName as "filterFileName".
working-storage section.
77 ret object reference "boolean".
linkage section.
01 f object reference j-io-file.
01 n object reference j-string.
procedure division using f, n
               returning ret.
main.
   if n:>toLowerCase:>endsWith(".cbl")
      set ret to true
   else
      set ret to false
   end-if
   goback.
end method.

end factory.

end class.
```

Sample application

The following is an example of a very simple banking application, consisting of one main program and the Account class. It is not intended to illustrate all the object oriented features of COBOL.

Main program

Most object oriented applications have a conventional program to start the processing. BANKMAIN serves this function in this sample bank application.

```
PROGRAM-ID. BANKMAIN.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
CLASS Account.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  an-object USAGE OBJECT REFERENCE Account.
PROCEDURE DIVISION.
go-now.
    INVOKE Account "new" RETURNING an-object.
    INVOKE an-object "displayUI".
    SET an-object to NULL.
    GOBACK.
```

The same code can also be written as follows:

```
PROGRAM-ID. BANKMAIN.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
CLASS Account.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  an-object USAGE OBJECT REFERENCE Account.
PROCEDURE DIVISION.
go-now.
    SET an-object TO Account::>new().
    an-object::>displayUI().
    SET an-object to NULL.
    GOBACK.
```

Account class

The source code for the account class is illustrated below. The class has two factory methods:

- addAccount adds 1 to the value of number-of-accounts, and
- removeAccount subtracts 1 from the value of number-of-accounts.

The account class also has six instance methods:

- newAccount creates a new instance of an account object,
- displayUI displays the value of the account balance or performs another function based on a user's request,
- balance retrieves the balance of the account,

- deposit adds an amount to the current balance of the account,
- withdraw subtracts an amount from the current balance of the account,
- initializeAccount moves initial values into the instance data.

```
CLASS-ID. Account.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
IDENTIFICATION DIVISION.
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  number-of-accounts     PIC 9(5) VALUE ZERO.
PROCEDURE DIVISION.

IDENTIFICATION DIVISION.
METHOD-ID. addAccount as "addAccount".
PROCEDURE DIVISION.
method-start.
    ADD 1 TO number-of-accounts.
END METHOD.

IDENTIFICATION DIVISION.
METHOD-ID. removeAccount as "removeAccount".
PROCEDURE DIVISION.
main-entry.
    SUBTRACT 1 FROM number-of-accounts.
END METHOD.
END FACTORY.

IDENTIFICATION DIVISION.
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  account-balance        PIC S9(9)V99.
01  account-number         PIC X(9).
01  the-date               PIC 9(8).
PROCEDURE DIVISION.

IDENTIFICATION DIVISION.
METHOD-ID. newAccount as "new".
DATA DIVISION.
PROCEDURE DIVISION.
begin-here.
    INVOKE SELF "initializeAccount" USING BY CONTENT number-of-accounts.
END METHOD.
```

```
IDENTIFICATION DIVISION.
METHOD-ID. displayUI as "displayUI".
DATA DIVISION.
WORKING-STORAGE SECTION.
01  in-data.
    03 action-type         PIC X.
    03 in-amount           PIC S9(9)V99.
    03 in-wrk              PIC X(12).
PROCEDURE DIVISION.
method-start.
    DISPLAY "Enter D for Deposit, B for Balance or W for Withdrawal"
    ACCEPT in-data
    EVALUATE action-type
    WHEN "D"
         PERFORM get-amount
         INVOKE SELF "deposit" USING in-amount
    WHEN "W"
         PERFORM get-amount
         INVOKE SELF "withdraw" USING in-amount
    WHEN "B"
         INVOKE SELF "balance"
    WHEN OTHER
         DISPLAY "Enter valid transaction type."
         GOBACK
    END-EVALUATE
    GOBACK
    .
get-amount.
    DISPLAY "Enter amount 9(9).99"
    ACCEPT in-wrk
    COMPUTE in-amount = FUNCTION NUMVAL (in-wrk)
    .
END METHOD.

IDENTIFICATION DIVISION.
METHOD-ID. balance as "balance".
DATA DIVISION.
WORKING-STORAGE SECTION.
01  display-balance        PIC $ZZZ,ZZZ,ZZ9.99.
PROCEDURE DIVISION.
disp-balance.
    MOVE account-balance to display-balance
    DISPLAY "Your Account Balance is:" display-balance
END METHOD.

IDENTIFICATION DIVISION.
METHOD-ID. deposit as "deposit".
DATA DIVISION.
LINKAGE SECTION.
01  in-deposit             PIC S9(9)V99.
PROCEDURE DIVISION USING in-deposit.
make-deposit.
    ADD in-deposit TO account-balance
END METHOD.
```

```
IDENTIFICATION DIVISION.
METHOD-ID. withdraw as "withdraw".
DATA DIVISION.
LINKAGE SECTION.
01  in-withdraw            PIC S9(9)V99.
PROCEDURE DIVISION USING in-withdraw.
withdraw-start.
    IF account-balance >= in-withdraw
       SUBTRACT in-withdraw FROM account-balance
    ELSE
       DISPLAY "Your Balance is Inadequate"
    END-IF
END METHOD.

IDENTIFICATION DIVISION.
METHOD-ID. initializeAccount as "initializeAccount".
DATA DIVISION.
LINKAGE SECTION.
01  new-account-number     PIC 9(5).
PROCEDURE DIVISION USING new-account-number.
Begin-initialization.
    MOVE ZERO TO account-balance
    MOVE new-account-number TO account-number
    ACCEPT the-date FROM CENTURY-DATE
END METHOD.
END OBJECT.
```

The following statements from the above code

```
INVOKE SELF "initializeAccount" USING BY CONTENT number-of-accounts
INVOKE SELF "deposit" USING in-amount
INVOKE SELF "withdraw" USING in-amount
INVOKE SELF "balance"
```

could also be written as

```
SELF:>initializeAccount (number-of-accounts)
SELF:>deposit (in-amount)
SELF:>withdraw (in-amount)
SELF:>balance ()
```

The main advantages in using the object:>method syntax instead of INVOKE are:

- the code is more similar to Java code, so it's easier to write down a cobol program that uses objects by converting an existing Java sample.
- more statements can appear in the same exception block. E.g., the following code

```
INVOKE myObj "method1"
  ON EXCEPTION
     DISPLAY MESSAGE exception-object:>getMessage()
END-INVOKE.
INVOKE myObj "method2"
  ON EXCEPTION
     DISPLAY MESSAGE exception-object:>getMessage()
END-INVOKE.
```

can be written as

```
TRY
   myObj:>method1();
   myObj:>method2();
CATCH EXCETPION
   DISPLAY MESSAGE exception-object:>getMessage()
END-TRY.
```

- Nested method invocation can be used. E.g., consider having an object named objParent, that provides a method called getChild. The getChild method returns and instance of objChild object, that has a method called doSomething. In order to use the doSomething method with INVOKE statement, two steps are required:

```
INVOKE objParent "getChild" GIVING objChild.
INVOKE objChild "doSomething".
```

With the object:>method syntax, instead, the objective can be achieved with one single statement:

```
objParent:>getChild:>doSomething().
```

Note - object:>method could be considered as a parameter of a previous COBOL statement if

- the previous COBOL statement supports multiple parameters (e.g. MOVE or DISPLAY)
- the previous COBOL statement isn't closed

If it's not possible to close the previous statement with a dot, use two semicolons.

For example, the following code is not accurate and leads to compiler errors and unwanted runtime behaviors:

```
if test-var = 1
   move 0 to test-var
   myObj:>myMethod1()
end-if
display test-var
myObj:>myMethod2()
```

myObj:>myMethod1 would be considered as a second parameter of the previous MOVE while the result of myObj:>myMethod2 would be displayed along with test-var.

The above code can be corrected as follows:

```
if test-var = 1
   move 0 to test-var;;
   myObj:>myMethod1()
end-if
display test-var.
myObj:>myMethod2()
```