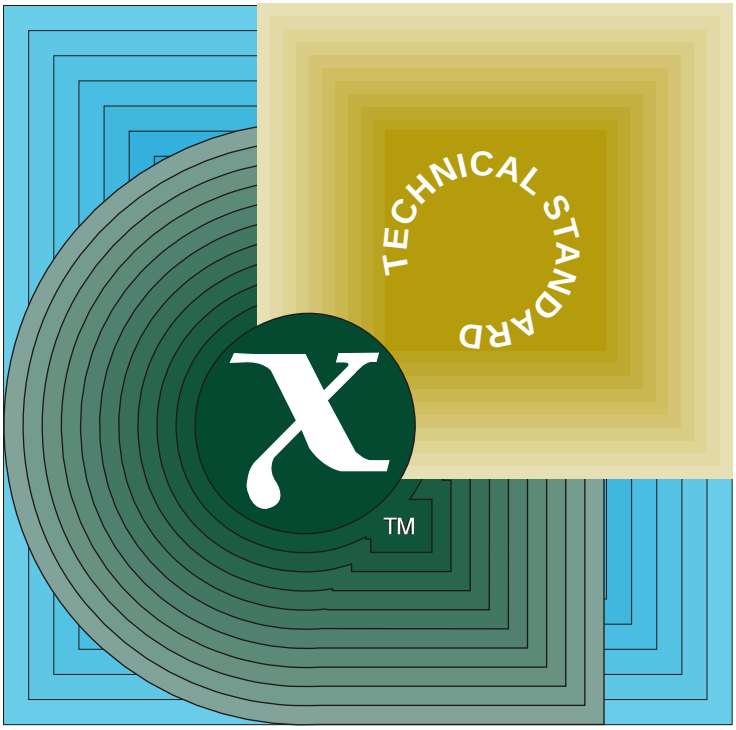
Technical Standard

COBOL Language



THE Open GROUP

[This page intentionally left blank]

X/Open CAE Specification

COBOL Language

X/Open Company, Ltd.

X/Open CAE Specification

COBOL Language

ISBN: 1 872630 09 X
X/Open Document Number: XO/CAE/91/200

Set in Palatino by X/Open Company Ltd., U.K.
Printed by Maple Press, U.K.
Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.co.uk

Contents

COBOL LANGUAGE

Preface

This Document

This document is a CAE Specification (see above). It provides the X/Open definition of the COBOL Language, which is that set of COBOL language facilities that programmers should follow when using COBOL compilers on X/Open-compliant systems.

The X/Open definition is based on the International Standard ISO 1989:1985 Programming Languages - COBOL (endorsement of ANSI Standard X3.23-1985). It contains all the required modules of the high level of Standard COBOL as well as the ISO 1989/Amendment 1 Intrinsic Function Module (endorsement of ANSI Standard X3.23a-1989). However, it contains none of the other optional modules of Standard COBOL. As several language items in ISO 1989:1985 are marked as obsolete, and will be removed in the next revision of Standard COBOL, these language items are not part of the X/Open definition.

This X/Open specification contains language enhancements that supplement Standard COBOL:

- the specification of a Screen Section and its use in ACCEPT and DISPLAY verbs for more complete interaction with the on-line user
- language extensions that allow sharing of files and locking of files and records
- a method to allow COBOL applications to use some of the internationalisation facilities defined in the X/Open **Basic Definitions** specification (see **Referenced Documents**)
- language extensions that enable multi-byte data to be handled in COBOL applications

Trademarks

X/Open and the 'X' device are trademarks of X/Open Company Limited in the U.K. and other countries.

UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and other countries.

Palatino is a trademark of Linotype AG and/or its subsidiaries.

Acknowledgements

X/Open gratefully acknowledges the assistance given by the following companies in the preparation of the COBOL definition:

- Liant Software Corporation
- Micro Focus Ltd.

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

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.

Referenced Documents

The following documents are referenced in this specification:

- Basic Definitions, Snapshot, X/Open Company Ltd., November 1991 (expected to be superseded by the CAE Specification, 1992)
- CODASYL COBOL Journal of Development, 1988
- Code of Japanese Graphic Character Set for Information Interchange (JIS KANJI FUGOUKEI, JIS X0208-1983)
- Indexed Sequential Access Method (ISAM), Developers' Specification, X/Open Company Ltd., 1990 (re-categorised as a CAE Specification)
- ISO 1989:1985 Programming Languages - COBOL (endorsement of ANSI Standard X3.23-1985)
- ISO 1989/Amendment 1 Intrinsic Function Module (endorsement of ANSI Standard X3.23a-1989)
- Multivendor Integration Architecture (MIA), Technical Requirements of the Application Program Interface Specification for the Programming Language COBOL, Version 1.0, February 1991
- Terminal Interfaces, CAE Specification, X/Open Company Ltd., 1992
- X/Open C Language definition contained in Programming Languages, CAE Specification, X/Open Company Ltd., 1992
- X/Open Portability Guide, Issue 1, 2 and 3

Chapter 1

Introduction

1.1 THE X/OPEN COBOL DEFINITION

The X/Open COBOL definition identifies a common set of language facilities that programmers should follow when using COBOL compilers on X/Open-compliant systems.

The X/Open definition is based on the International Standard ISO 1989:1985 Programming Languages - COBOL (endorsement of ANSI Standard X3.23-1985). It contains all the required modules of the high level of Standard COBOL as well as the ISO 1989/Amendment 1 Intrinsic Function Module (endorsement of ANSI Standard X3.23a-1989). Both standards are commonly referred to as ''Standard COBOL'' throughout this specification. However, it contains none of the other optional modules of Standard COBOL. As several language items in ISO 1989:1985 are marked as obsolete, and will be removed in the next revision of Standard COBOL, these language items are not part of the X/Open definition. Although implementations are required to support these features, application writers should avoid using them.

Standard COBOL is incomplete in the area of facilities for interaction with the on-line user. This X/Open specification defines a Screen Section and its use in ACCEPT and DISPLAY verbs, as this is the most common language realisation among COBOL compilers. Standard COBOL does not define facilities for efficient and consistent file sharing in a multi-user environment. This X/Open specification defines language extensions that allow sharing of files and locking of files and records. This X/Open specification furthermore defines a method to allow COBOL applications to use the internationalisation facilities defined in the X/Open **Basic Definitions** specification (see **Referenced Documents**). Finally enhancements to the COBOL language are defined that enable multi-byte data to be handled in COBOL applications.

As X/Open specifies more or less a common subset of the languages of various compiler products, there are extensions supported on specific X/Open-compliant systems that are not included in the X/Open definition. Also the means of implementation across X/Open-compliant systems may vary.

For certain elements in the implementor-defined element list in Standard COBOL, X/Open defines a standard implementation across all X/Open-compliant systems. Where this has been done, the clause is annotated in the syntax definition.

The X/Open COBOL definition is given in **Chapter 2**, **COBOL Definition**. It is derived from the Syntax Summary of Standard COBOL plus extensions taken from common implementations. The semantics of the language are those of Standard COBOL. In the general formats the facilities that are additional to Standard COBOL, and included in the X/Open definition, are indicated by shading. With the exception of those extensions that are fully defined by their syntax, a full definition of their semantics is contained in **Chapter 3**, **Definition of Extensions**. The major extensions mentioned above are contained in **Chapter 4**, **Screen Handling Module**, **Chapter 5**, **File Sharing and Record Locking** and **Chapter 6**, **Internationalisation**.

Chapter 7, **Portability Issues** summarises the functions in Standard COBOL that are not to be used in a conforming X/Open COBOL source program. These are described in relation to the ''modules'' defined in Standard COBOL. This information is included to allow those familiar with Standard COBOL to obtain a quick appreciation of the X/Open definition.

Chapter 8, **COBOL in an X/Open Environment** addresses the special characteristics of X/Open-compliant systems and the effects these have on the functionality of COBOL compilers, with recommended techniques for ensuring the maximum portability of COBOL programs between X/Open-compliant implementations.

A conforming implementation of X/Open COBOL shall fully support any of the three subsets of ISO 1989:1985 (including ISO 1989/Amendment 1), and additionally the standard extensions defined in **Chapter 3**, **Definition of Extensions**, **Chapter 4**, **Screen Handling Module**, **Chapter 5**, **File Sharing and Record Locking**, **Chapter 6**, **Internationalisation** and **Chapter 8**, **COBOL in an X/Open Environment**. Elements of the implementor-defined element list in Standard COBOL that are defined by X/Open have to be supported in the X/Open way. However, X/Open-compliant implementations only need to provide support for handling of multi-byte character sets if the environment provides support for it. Elements of Standard COBOL that are not mentioned shall be supported by a conforming implementation, but are not guaranteed to be portable.

A conforming X/Open COBOL source program is a program that only uses language features of the high subset of Standard COBOL (including ISO 1989/Amendment 1) or those defined in **Chapter 3**, **Definition of Extensions**, **Chapter 4**, **Screen Handling Module**, **Chapter 5**, **File Sharing and Record Locking** and **Chapter 6**, **Internationalisation**, and does not use language features of **Chapter 7**, **Portability Issues**. However, the obsolete language features specified in this chapter are permitted in conforming source programs. Where, in **Chapter 8**, **COBOL in an X/Open Environment**, further definitions are made concerning the X/Open environment or elements in the implementor-defined element list, the program should conform to these. However, X/Open-compliant applications should not use internationalisation support or multi-byte character facilities if the environment does not provide support for it.

1.2 HISTORY OF THIS DOCUMENT

The **X/Open Portability Guide**, **Issue 1** and **Issue 2** are based on the ANSI Standard X3.23-1974 with extensions for interaction with the on-line user. The **X/Open Portability Guide**, **Issue 3** is based on ISO 1989:1985 Programming Languages - COBOL (endorsement of ANSI Standard X3.23-1985) with a changed extension for interaction with the on-line user. In addition, other minor extensions to Standard COBOL are defined.

This specification continues alignment with Standard COBOL by including the ISO 1989/Amendment 1 Intrinsic Function Module (endorsement of ANSI Standard X3.23a-1989). Moreover, it contains the following additional extensions:

- file sharing and record locking (replacing the implementation-dependent option defined in the **X/Open Portability Guide**, **Issue 3**)
- support for internationalisation facilities
- support for multi-byte character sets
- free form reference format (in order to facilitate handling of multi-byte character sets)
- concatenation of nonnumeric and national-character literals (in order to facilitate handling of multi-byte character sets)
- passing parameters by value (in order to facilitate interaction with other components of the Common Applications Environment (CAE))

In this CAE Specification, the continuation of COBOL words, PICTURE character strings and literals is an obsolete element because it will be deleted in a future edition, i.e., application writers should avoid using this feature.

1.3 FORMAT OF ENTRIES

The clauses and statements described in **Chapter 3**, **Definition of Extensions**, **Chapter 4**, **Screen Handling Module**, **Chapter 5**, **File Sharing and Record Locking**, **Chapter 6**, **Internationalisation** and **Chapter 8**, **COBOL in an X/Open Environment** are based on a common format, aligned with that of Standard COBOL.

Function States briefly the purpose of the clause or statement.

General Format Summarises the use of the clause or statement being described. It is the specific arrangement of its elements. When more than one specific arrangement is permitted that need to be referred to specifically, the general format is separated into numbered formats.

Syntax Rules Defines or clarifies the order in which words or elements are arranged to form larger elements such as phrases, clauses or statements. Syntax rules may also either impose restrictions on individual words or elements, or relax restrictions implied by words, elements or Standard COBOL.

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

General Rules Defines or clarifies the meaning, or relationship of meanings, of an element or set of elements. It is used to define or clarify the semantics of the statement, and the effect that it has on either execution or compilation.

Examples Gives examples of usage, where appropriate.

Application Usage
Gives warnings and advice to application writers about the clause or statement.

The formal description consists only of Function, General Format, Syntax Rules and General Rules sections.

1.3.1 Typographical Conventions

The General Formats use Standard COBOL notation to define the language syntax.

Upper case COBOL language keywords and implementation-dependent names which are defined by X/Open. Those underlined must be present if the clause is present; those not underlined are optional words, which may be included to improve readability but are otherwise not processed by the compiler.

Lower case Generic terms which represent substitutable arguments, for example, data-names and literal values.

Square brackets Used to enclose optional alternatives; any clause not so enclosed is mandatory. One or none of the alternatives enclosed within the square brackets may be selected. Square brackets may also be used together with ellipses to delimit a repeatable construct.

Braces Used to enclose alternatives. One of the alternatives enclosed within the braces must be used. Braces may also be used together with ellipses to delimit a repeatable construct.

Ellipses The preceding element may be repeated a number of times. There must be at least one occurrence, unless the element is optional (enclosed in square brackets).

Choice indicators {| and |}
Used to enclose alternatives. One or more of the unique options contained within the choice indicators must be used, but a single option may be specified only once.

Shaded areas Areas where the X/Open definition extends or clarifies Standard COBOL.

The reason for the shading is indicated in the right margin by:

E extension included in the definition

I an element which is in the implementor-defined element list in Standard COBOL, but which is defined by X/Open

1.3.2 Terminology

The following terms are used in this specification:

implementation-dependent
The feature is not consistent across all implementations, and each implementation shall provide documentation of its behaviour. Implementation-dependent elements shall conform to the rules set by Standard COBOL.

may With respect to implementations, the feature is optional. Applications should not rely on its existence.

With respect to applications, the word is used to give guidelines for recommended practice if the described functionality is desired in the application. These guidelines should be followed if maximum portability is desired.

must The rule must be obeyed by conforming applications.

should With respect to implementations, the feature is recommended, but it is not a mandatory requirement. Applications should not rely on the existence of the feature.

With respect to applications, the word is used to give guidelines for recommended practice. These guidelines should be followed if maximum portability is desired.

undefined A feature is undefined if this specification imposes no portability requirements on applications for erroneous program constructs or erroneous data. Implementations may specify the result of using the feature, but such specifications are not guaranteed to be consistent across all implementations. However, undefined elements shall

conform to the rules set by Standard COBOL.

unspecified A feature is unspecified if this specification imposes no portability requirements on applications for correct program constructs or correct data. Implementations may specify the result of using the feature, but such specifications are not guaranteed to be consistent across all implementations. However, unspecified elements shall conform to the rules set by Standard COBOL.

shall The feature must be implemented and applications can rely on its existence.

Chapter 2

COBOL Definition

The definition is derived from Section V (Composite Language Skeleton) of ISO 1989:1985, together with the changes to this section from ISO 1989/Amendment 1 (Changes to Standard COBOL).

2.1 GENERAL FORMAT FOR A SEQUENCE OF SOURCE PROGRAMS

```
{ IDENTIFICATION DIVISION .
PROGRAM-ID . program-name-1 [ IS INITIAL PROGRAM ] .
[ ENVIRONMENT DIVISION . [ environment-division-content ] ]
[ DATA DIVISION . [ data-division-content ] ]
[ PROCEDURE DIVISION . procedure-division-content ]
[ nested-source-program ] . . .
END PROGRAM program-name-1 . } . . .
IDENTIFICATION DIVISION .
PROGRAM-ID . program-name-2 [ IS INITIAL PROGRAM ] .
[ ENVIRONMENT DIVISION . [ environment-division-content ] ]
[ DATA DIVISION . [ data-division-content ] ]
[ PROCEDURE DIVISION . procedure-division-content ]
[ [ nested-source-program ] . . .
END PROGRAM program-name-2 . ]
```

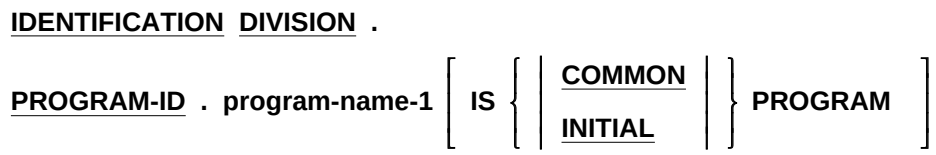
2.2 GENERAL FORMAT FOR NESTED SOURCE PROGRAMS

```
IDENTIFICATION DIVISION .
PROGRAM-ID . program-name-1 [ IS INITIAL PROGRAM ] .
[ ENVIRONMENT DIVISION . [ environment-division-content ] ]
[ DATA DIVISION . [ data-division-content ] ]
[ PROCEDURE DIVISION . procedure-division-content ]
[ [ nested-source-program ] . . .
END PROGRAM program-name-1 . ]
```

2.2.1 Nested-Source-Program

```
IDENTIFICATION DIVISION .
PROGRAM-ID . program-name-1 [ IS { COMMON  } PROGRAM ] .
                                 { INITIAL }
[ ENVIRONMENT DIVISION . [ environment-division-content ] ]
[ DATA DIVISION . [ data-division-content ] ]
[ PROCEDURE DIVISION . procedure-division-content ]
[ nested-source-program ] . . .
END PROGRAM program-name-1 .
```

2.3 GENERAL FORMAT FOR IDENTIFICATION DIVISION



2.4 GENERAL FORMAT FOR ENVIRONMENT DIVISION

```
ENVIRONMENT DIVISION .
[ CONFIGURATION SECTION .
[ SOURCE-COMPUTER . [ source-computer-entry ] ]
[ OBJECT-COMPUTER . [ object-computer-entry ] ]
[ SPECIAL-NAMES . [ special-names-contents ] ] ]
[ INPUT-OUTPUT SECTION .
FILE-CONTROL . { file-control-entry } . . .
[ I-O-CONTROL . [ same-area-entry ] ] ]
```

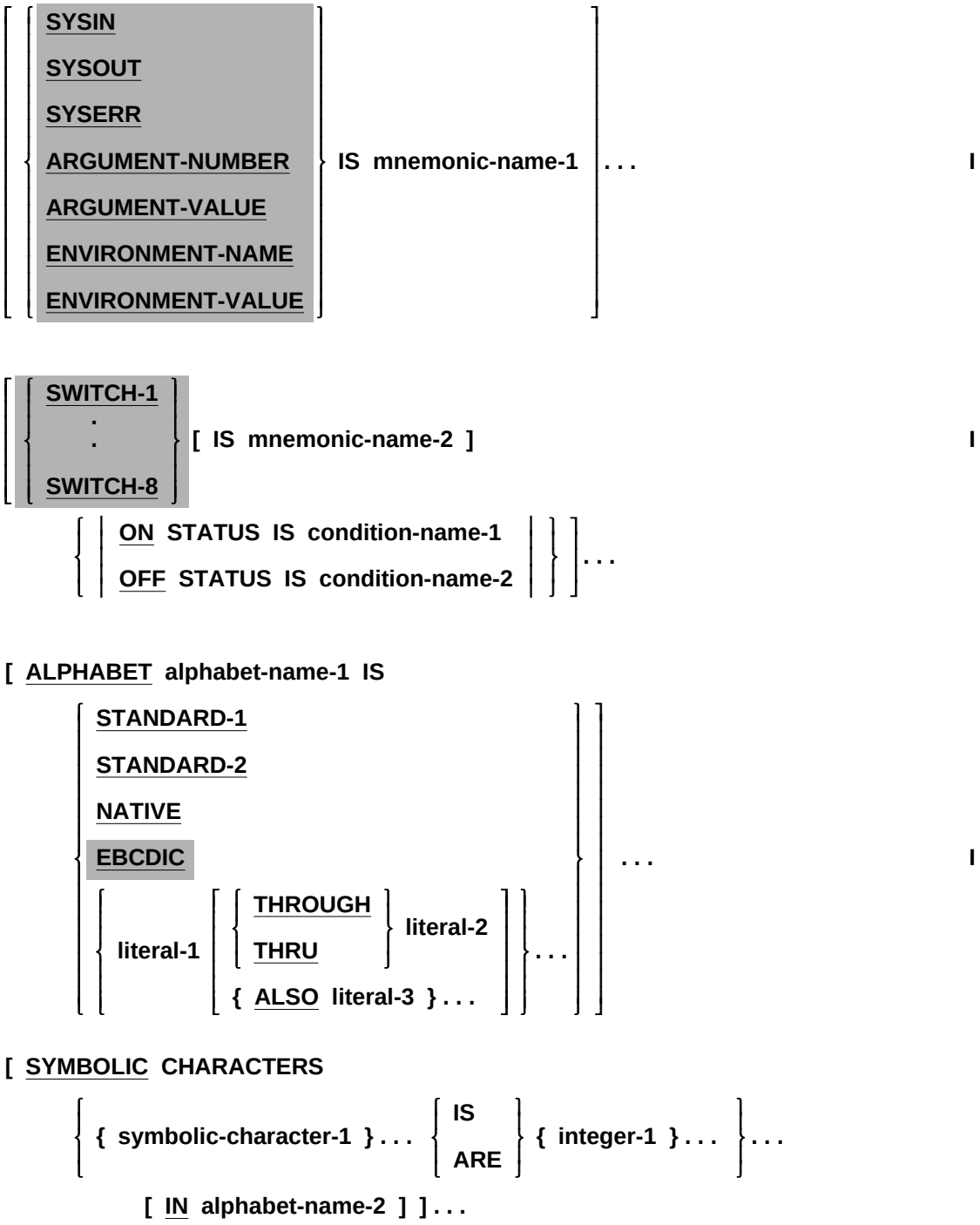
2.5 GENERAL FORMAT FOR SOURCE-COMPUTER-ENTRY

computer-name-1 [WITH DEBUGGING MODE] .

2.6 GENERAL FORMAT FOR OBJECT-COMPUTER-ENTRY

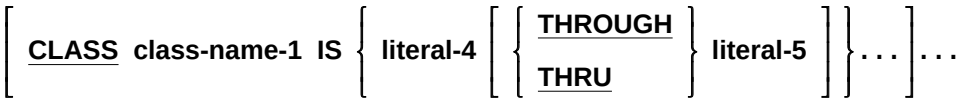
computer-name-1 [PROGRAM COLLATING SEQUENCE IS alphabet-name-1] .

2.7 GENERAL FORMAT FOR SPECIAL-NAMES-CONTENT



(continued overleaf)

General Format for Special-Names-Content (continued)



[CURRENCY SIGN IS literal-6]

[DECIMAL-POINT IS COMMA]

[CURSOR IS data-name-1] E

[CRT STATUS IS data-name-2] . E

2.8 GENERAL FORMAT FOR FILE-CONTROL-ENTRY

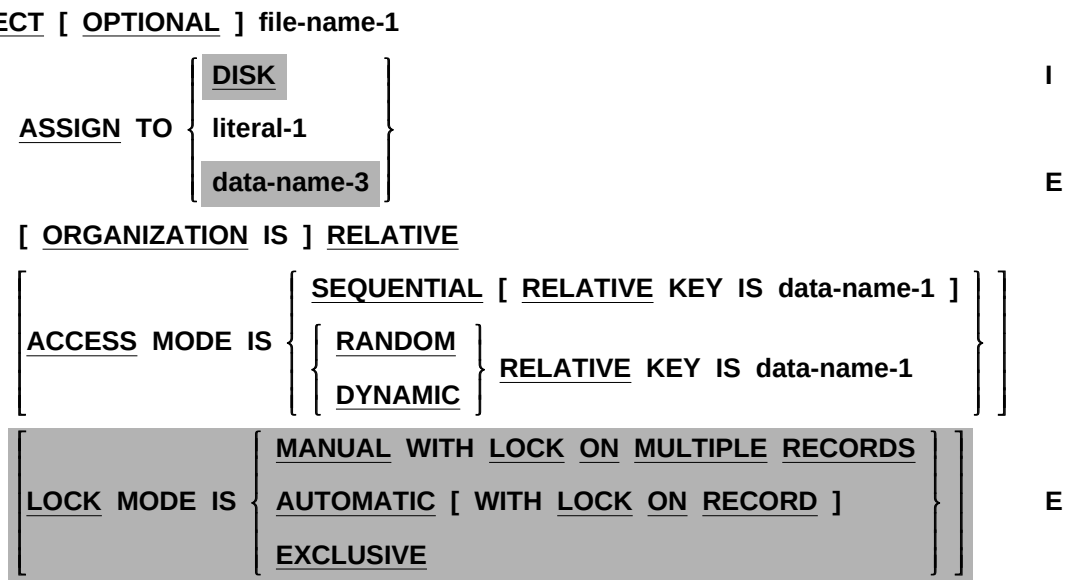
2.8.1 Sequential File

```
SELECT [ OPTIONAL ] file-name-1
        ASSIGN TO { DISK          }                                   I
                  { PRINTER       }
                  { literal-1     }
                  { data-name-1   }                                   E
    [ [ ORGANIZATION IS ] SEQUENTIAL ]
    [ ACCESS MODE IS SEQUENTIAL ]
    [ LOCK MODE IS { AUTOMATIC [ WITH LOCK ON RECORD ] } ]            E
                   { EXCLUSIVE                         }
    [ FILE STATUS IS data-name-2 ] .
```

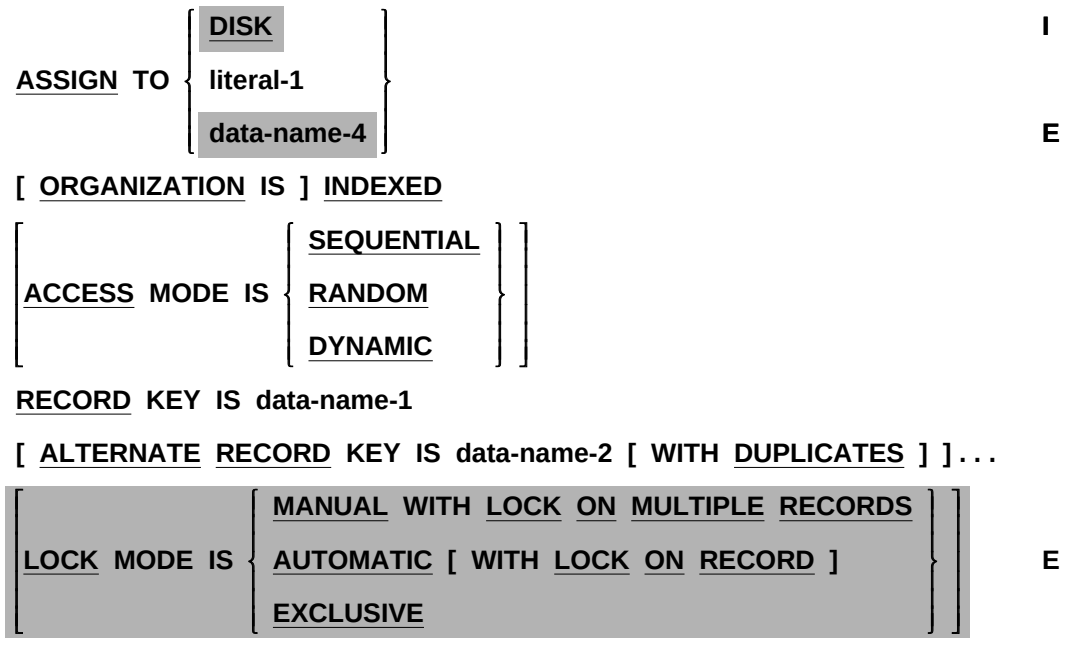
2.8.2 Line Sequential File

```
SELECT [ OPTIONAL ] file-name-1
        ASSIGN TO { DISK        }
                  { literal-1   }
                  { data-name-1 }
    [ ORGANIZATION IS ] LINE SEQUENTIAL                               E
    [ ACCESS MODE IS SEQUENTIAL ]
    [ LOCK MODE IS { AUTOMATIC [ WITH LOCK ON RECORD ] } ]
                   { EXCLUSIVE                         }
    [ FILE STATUS IS data-name-2 ] .
```

2.8.3 Relative File



2.8.4 Indexed File



2.8.5 Sort or Merge File

2.9 GENERAL FORMAT FOR SAME-AREA-ENTRY

2.10 GENERAL FORMAT FOR DATA DIVISION

```
DATA DIVISION .

[ FILE SECTION .
[ file-description-entry { record-description-entry }... ]...
[ sort-merge-file-description-entry { record-description-entry }... ]... ]

[ WORKING-STORAGE SECTION .
[ 77-level-description-entry ]...
[ record-description-entry ]... ]

[ LINKAGE SECTION .
[ 77-level-description-entry ]...
[ record-description-entry ]... ]

[ SCREEN SECTION .                                                    E
[ screen-description-entry ]... ]
```

2.11 GENERAL FORMAT FOR FILE-DESCRIPTION-ENTRY

2.11.1 Sequential or Line Sequential File

```
FD file-name-1
    [ IS EXTERNAL ]
    [ IS GLOBAL ]
    [ RECORD { CONTAINS integer-3 CHARACTERS                                  } ]
             { IS VARYING IN SIZE                                             }
             {     [ [ FROM integer-4 ] [ TO integer-5 ] CHARACTERS ]         }
             {     [ DEPENDING ON data-name-1 ]                               }
    [ LINAGE IS { data-name-4 } LINES                                         ]
                { integer-8   }
        [ WITH FOOTING AT { data-name-5 } ]
                          { integer-9   }
        [ LINES AT TOP { data-name-6 } ]
                       { integer-10  }
        [ LINES AT BOTTOM { data-name-7 } ] ] .
                          { integer-11  }
```

2.11.2 Relative or Indexed File

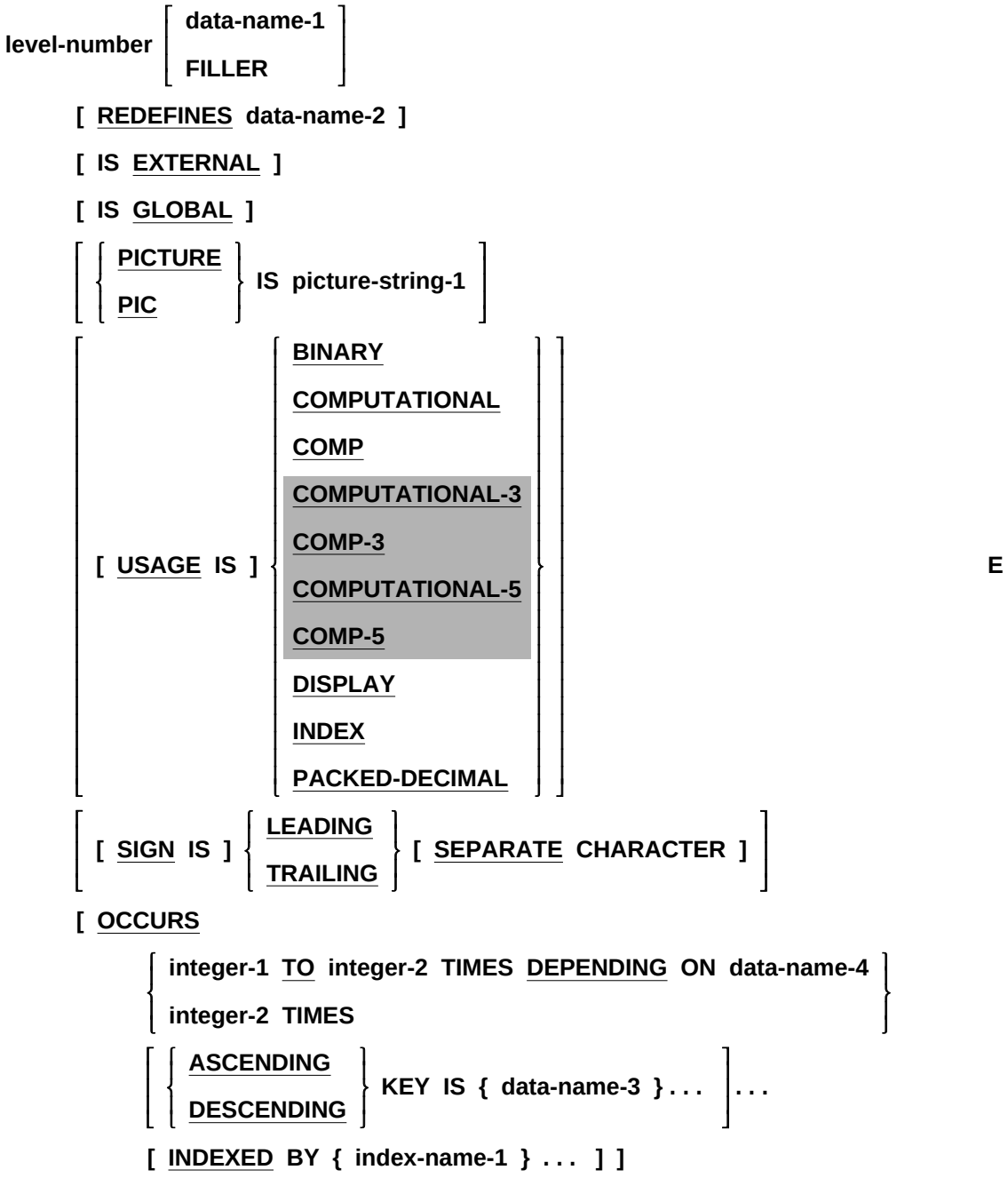
```
FD file-name-1
    [ IS EXTERNAL ]
    [ IS GLOBAL ]
    [ RECORD { CONTAINS integer-3 CHARACTERS                                  } ] .
             { IS VARYING IN SIZE                                             }
             {     [ [ FROM integer-4 ] [ TO integer-5 ] CHARACTERS ]         }
             {     [ DEPENDING ON data-name-1 ]                               }
```

2.11.3 Sort or Merge File

```
SD file-name-1
   [ RECORD { CONTAINS integer-1 CHARACTERS
            { IS VARYING IN SIZE
                 [ [ FROM integer-2 ] [ TO integer-3 ] CHARACTERS ]
                 [ DEPENDING ON data-name-1 ]                        } ] .
```

2.12 GENERAL FORMAT FOR DATA DESCRIPTION ENTRY

2.12.1 Format 1



(continued overleaf)

Format 1 (continued)

```
[ { JUSTIFIED } RIGHT ]
  { JUST      }
[ BLANK WHEN ZERO ]
[ VALUE IS literal-1 ] .
```

2.12.2 Format 2

```
66 data-name-1 RENAMES data-name-2 [ { THROUGH } data-name-3 ] .
                                   { THRU    }
```

2.12.3 Format 3

```
88 condition-name-1 { VALUE IS  } { literal-1 [ { THROUGH } literal-2 ] }... .
                    { VALUES ARE }            { THRU    }
```

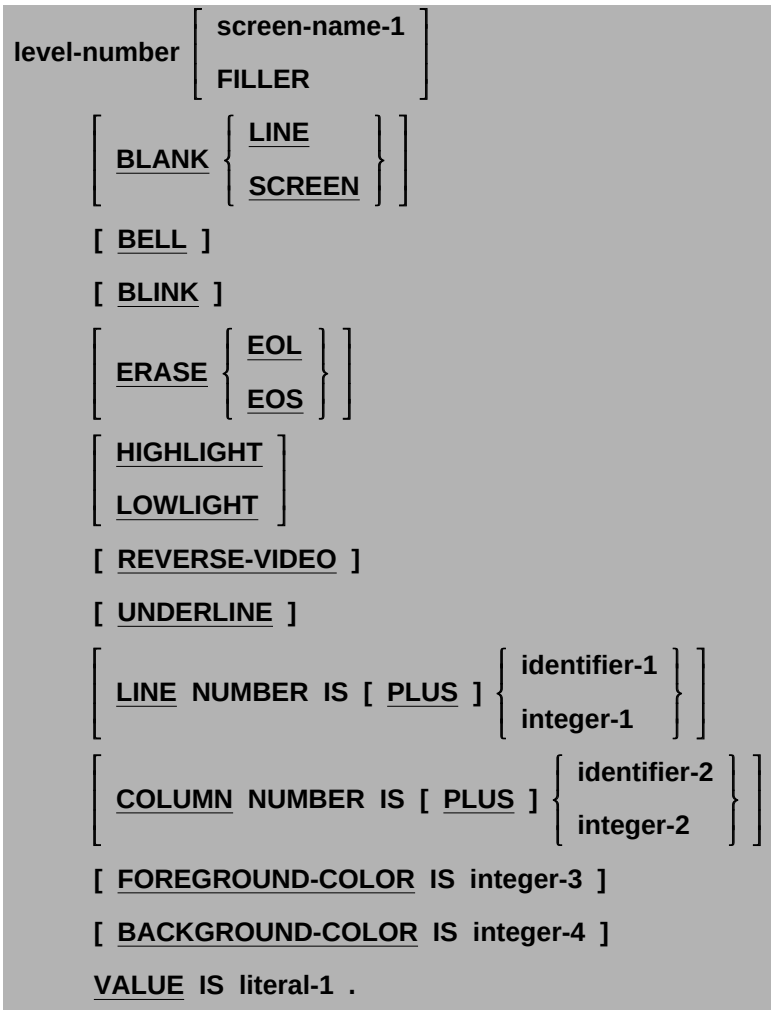
2.13 GENERAL FORMAT FOR SCREEN-DESCRIPTION-ENTRY

2.13.1 Format 1

```
level-number [ screen-name-1 ]
             [ FILLER        ]
      [ BLANK SCREEN ]
      [ FOREGROUND-COLOR IS integer-3 ]
      [ BACKGROUND-COLOR IS integer-4 ]
      [ AUTO ]
      [ SECURE ]
      [ REQUIRED ]
      [ [ USAGE IS ] DISPLAY ]
      [ [ SIGN IS ] { LEADING  } [ SEPARATE CHARACTER ] ]
                    { TRAILING }
      [ FULL ] .
```

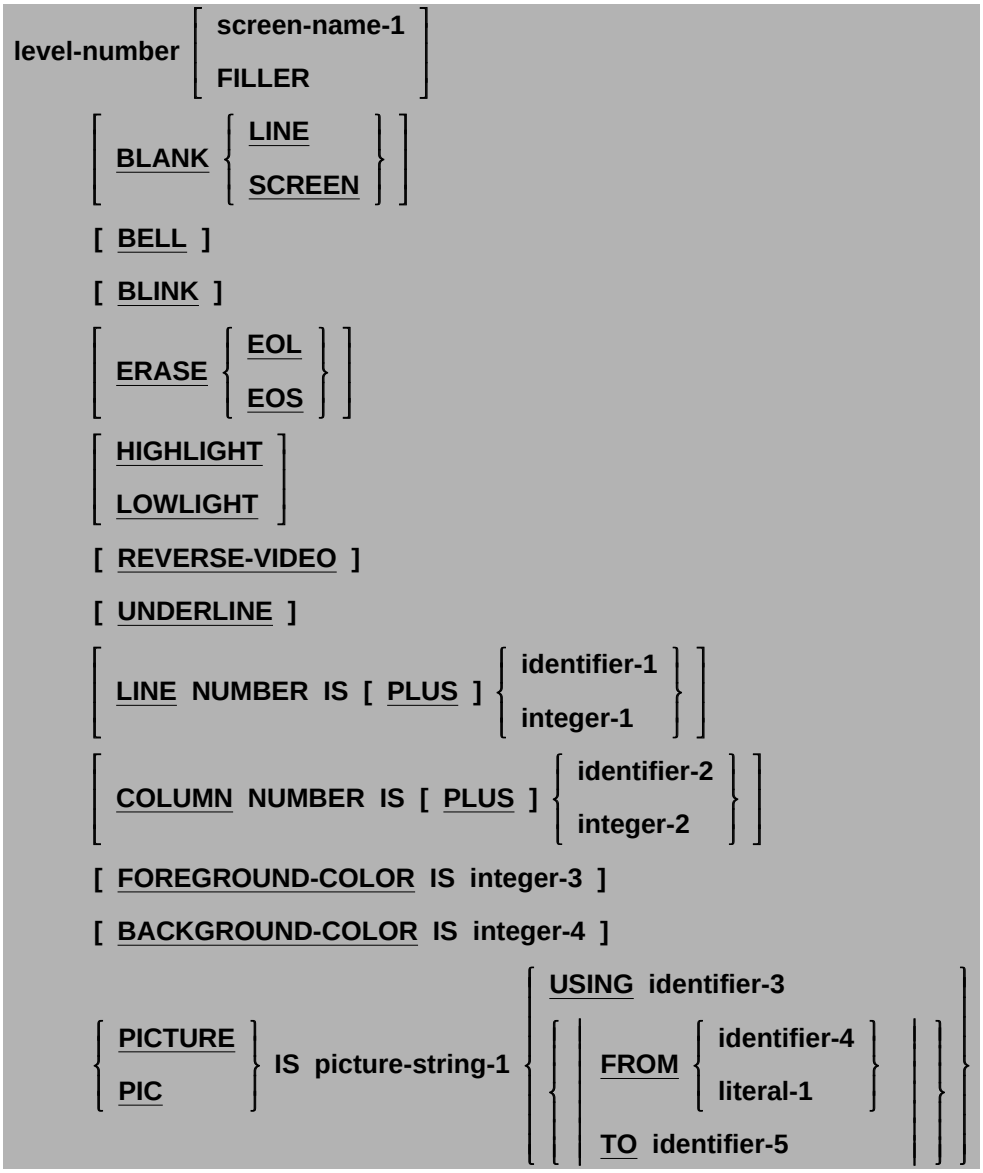
E

2.13.2 Format 2



E

2.13.3 Format 3



E

(continued overleaf)

Format 3 (continued)

```
[ [ USAGE IS ] DISPLAY ]
[ BLANK WHEN ZERO ]
[ { JUSTIFIED } RIGHT ]
[ { JUST      }       ]
[ [ SIGN IS ] { LEADING  } [ SEPARATE CHARACTER ] ]
              { TRAILING }
[ AUTO ]
[ SECURE ]
[ REQUIRED ]
[ FULL ] .
```

E

2.14 GENERAL FORMAT FOR PROCEDURE DIVISION

2.14.1 Format 1

```
PROCEDURE DIVISION [ USING { data-name-1 } ... ] .
[ DECLARATIVES .
{ section-name-1 SECTION . USE statement .
[ paragraph-name-1 . [ sentence ] ... ] ... } ...
END DECLARATIVES . ]
{ section-name-2 SECTION .
[ paragraph-name-2 . [ sentence ] ... ] ... } ...
```

2.14.2 Format 2

```
PROCEDURE DIVISION [ USING { data-name-1 } ... ] .
{ paragraph-name-1 . [ sentence ] ... } ...
```

2.15 GENERAL FORMAT FOR ACCEPT STATEMENT

```
ACCEPT identifier-1 [ FROM mnemonic-name-1 ] [ END-ACCEPT ]
```

E

```
ACCEPT identifier-1 FROM mnemonic-name-1
    [ ON EXCEPTION imperative-statement-1 ]
    [ NOT ON EXCEPTION imperative-statement-2 ]
    [ END-ACCEPT ]
```

E

```
ACCEPT screen-name-1
    [ AT { { LINE NUMBER { identifier-1 } } [ { identifier-2 } ] }
         { { COLUMN NUMBER { integer-1  } } [ { integer-2  } ] } ]
    [ ON EXCEPTION imperative-statement-1 ]
    [ NOT ON EXCEPTION imperative-statement-2 ]
    [ END-ACCEPT ]
```

E

```
ACCEPT identifier-2 FROM { DATE        } [ END-ACCEPT ]
                         { DAY         }
                         { DAY-OF-WEEK }
                         { TIME        }
```

E

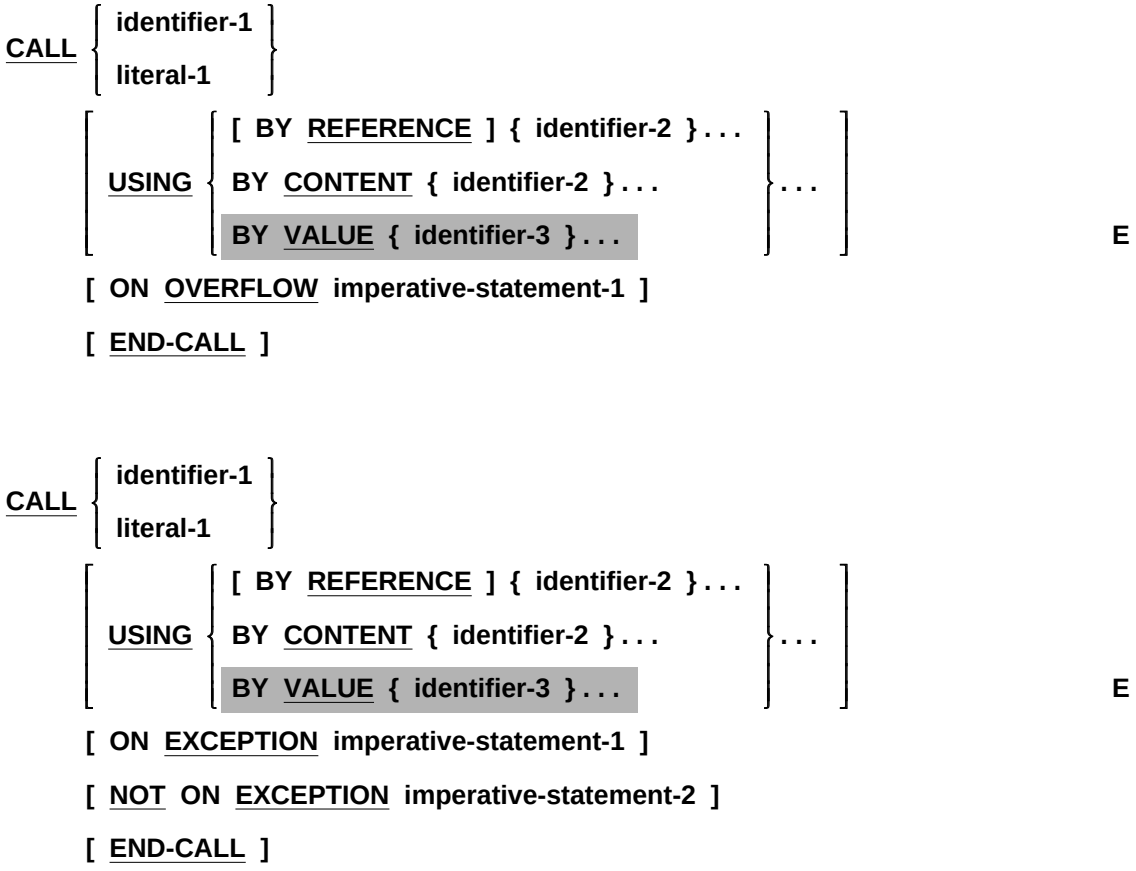
2.16 GENERAL FORMAT FOR ADD STATEMENT

```
ADD { identifier-1 }...  TO { identifier-2 [ ROUNDED ] }...
    { literal-1    }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-ADD ]
```

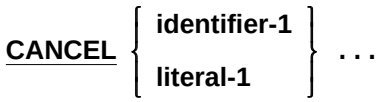
```
ADD { identifier-1 }...  TO { identifier-2 }
    { literal-1    }        { literal-2    }
    GIVING { identifier-3 [ ROUNDED ] }...
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-ADD ]
```

```
ADD { CORRESPONDING } identifier-1 TO identifier-2 [ ROUNDED ]
    { CORR          }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-ADD ]
```

2.17 GENERAL FORMAT FOR CALL STATEMENT

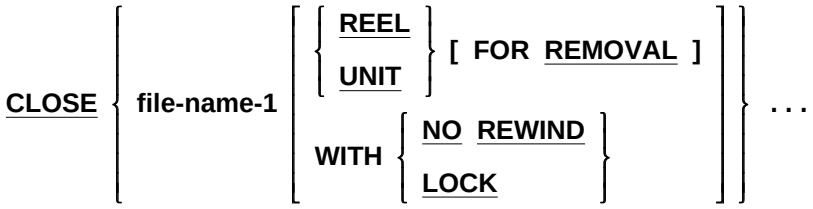


2.18 GENERAL FORMAT FOR CANCEL STATEMENT



2.19 GENERAL FORMAT FOR CLOSE STATEMENT

2.19.1 Sequential or Line Sequential File



2.19.2 Relative or Indexed File

```
CLOSE { file-name-1 [ WITH LOCK ] } . . .
```

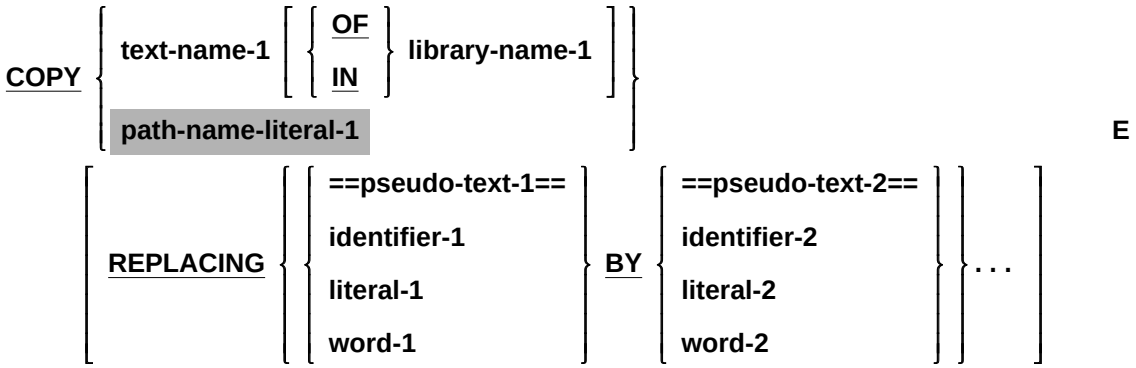
2.20 GENERAL FORMAT FOR COMPUTE STATEMENT

```
COMPUTE { identifier-1 [ ROUNDED ] } . . . = arithmetic-expression-1
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-COMPUTE ]
```

2.21 GENERAL FORMAT FOR CONTINUE STATEMENT

CONTINUE

2.22 GENERAL FORMAT FOR COPY STATEMENT



2.23 GENERAL FORMAT FOR DELETE STATEMENT

```
DELETE file-name-1 RECORD
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-DELETE ]
```

2.24 GENERAL FORMAT FOR DISPLAY STATEMENT

```
DISPLAY { identifier-1 }...  [ UPON mnemonic-name-1 ]
        { literal-1    }
     [ WITH NO ADVANCING ]
     [ END-DISPLAY ]                                          E
```

```
DISPLAY { identifier-1 }...  UPON mnemonic-name-1
        { literal-1    }
     [ ON EXCEPTION imperative-statement-1 ]                  E
     [ NOT ON EXCEPTION imperative-statement-2 ]
     [ END-DISPLAY ]
```

```
DISPLAY screen-name-1
     [ AT { | LINE NUMBER { identifier-1 } | }                E
          { |             { integer-1    } | }
          { | COLUMN NUMBER { identifier-2 } | }
          { |               { integer-2    } | } ]
     [ END-DISPLAY ]
```

2.25 GENERAL FORMAT FOR DIVIDE STATEMENT

```
DIVIDE { identifier-1 } INTO { identifier-2 [ ROUNDED ] } . . .
       { literal-1    }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-DIVIDE ]
```

```
DIVIDE { identifier-1 } { INTO } { identifier-2 }
       { literal-1    } { BY   } { literal-2    }
    GIVING { identifier-3 [ ROUNDED ] } . . .
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-DIVIDE ]
```

```
DIVIDE { identifier-1 } { INTO } { identifier-2 }
       { literal-1    } { BY   } { literal-2    }
    GIVING identifier-3 [ ROUNDED ]
    REMAINDER identifier-4
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-DIVIDE ]
```

2.26 GENERAL FORMAT FOR EVALUATE STATEMENT

```
EVALUATE selection-subject-1 [ ALSO selection-subject-2 ] . . .
    { { WHEN selection-object-1 [ ALSO selection-object-2 ] . . . } . . .
        imperative-statement-1 } . . .
    [ WHEN OTHER imperative-statement-2 ]
    [ END-EVALUATE ]
```

Selection-Subject

```
{ identifier-1      }
{ literal-1         }
{ expression-1      }
{ TRUE              }
{ FALSE             }
```

Selection-Object

```
{ ANY                                                                          }
{ condition-1                                                                  }
{ TRUE                                                                         }
{ FALSE                                                                        }
{ [ NOT ] { { identifier-2            }                                        }
{         { { literal-2               }                                        }
{         { { arithmetic-expression-1 }                                        }
{           [ { THROUGH } { identifier-2            } ]                        }
{           [ {         } { literal-2               } ]                        }
{           [ { THRU    } { arithmetic-expression-2 } ]                        }
```

2.27 GENERAL FORMAT FOR EXIT STATEMENT

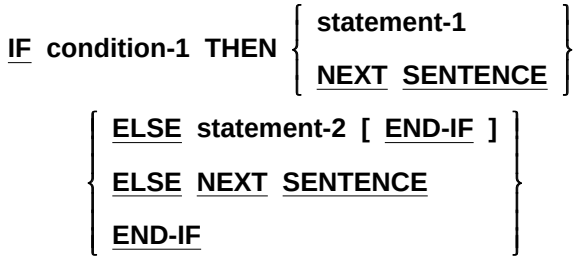
EXIT [PROGRAM]

2.28 GENERAL FORMAT FOR GO TO STATEMENT

GO TO procedure-name-1

GO TO { procedure-name-1 } . . . DEPENDING ON identifier-1

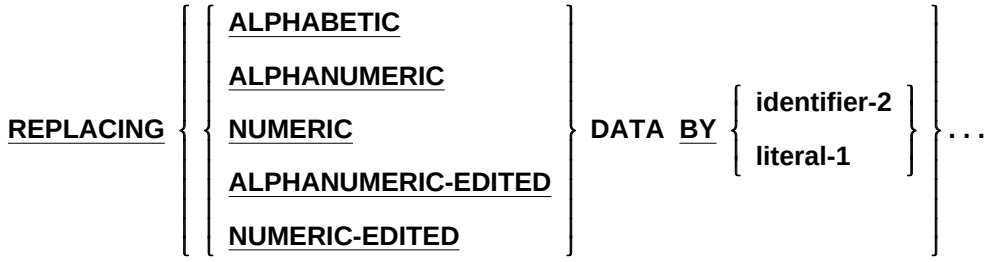
2.29 GENERAL FORMAT FOR IF STATEMENT



2.30 GENERAL FORMAT FOR INITIALIZE STATEMENT

```
INITIALIZE { identifier-1 } . . . [ replacing-phrase ]
```

Replacing-Phrase



2.31 GENERAL FORMAT FOR INSPECT STATEMENT

```
INSPECT identifier-1 TALLYING { tally-phrase } ...

INSPECT identifier-1 REPLACING { replacing-phrase } ...

INSPECT identifier-1 TALLYING { tally-phrase } ...
        REPLACING { replacing-phrase } ...

INSPECT identifier-1 CONVERTING { identifier-6 | literal-4 } TO { identifier-7 | literal-5 }
        [ before-after-phrase ] ...
```

Tally-Phrase

```
identifier-2 FOR { CHARACTERS [ before-after-phrase ] ...
                 | { ALL | LEADING } { { identifier-3 | literal-1 } [ before-after-phrase ] ... }... }...
```

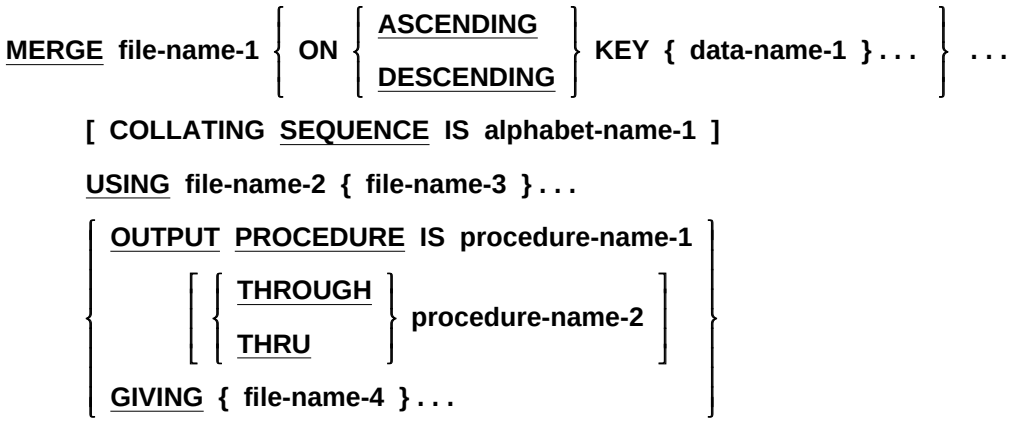
Replacing-Phrase

```
{ CHARACTERS BY { identifier-5 | literal-3 } [ before-after-phrase ] ...
| { ALL | LEADING | FIRST } { { identifier-3 | literal-1 } BY { identifier-5 | literal-3 } [ before-after-phrase ] ... }... }
```

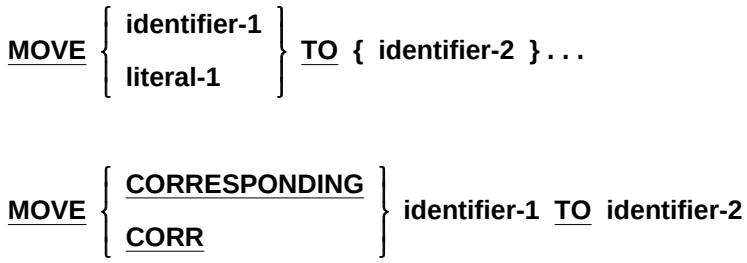
Before-After-Phrase

```
{ BEFORE | AFTER } INITIAL { identifier-4 | literal-2 }
```

2.32 GENERAL FORMAT FOR MERGE STATEMENT



2.33 GENERAL FORMAT FOR MOVE STATEMENT



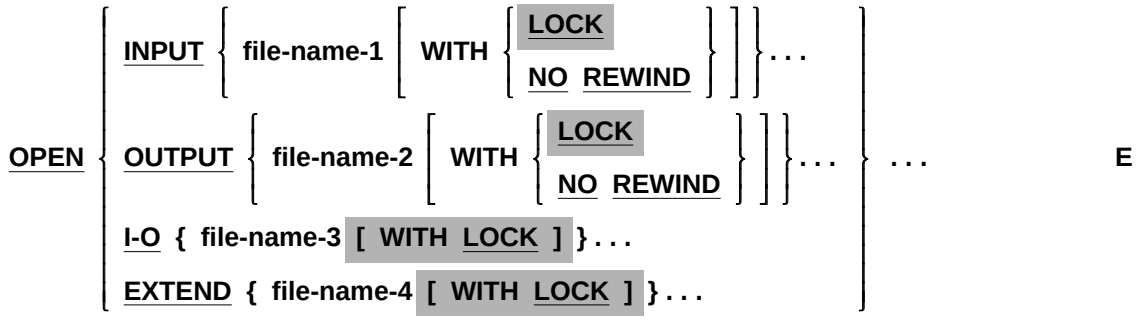
2.34 GENERAL FORMAT FOR MULTIPLY STATEMENT

```
MULTIPLY { identifier-1 } BY { identifier-2 [ ROUNDED ] } . . .
         { literal-1    }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-MULTIPLY ]
```

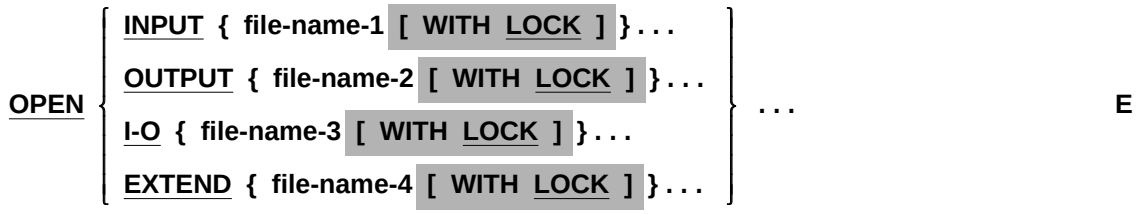
```
MULTIPLY { identifier-1 } BY { identifier-2 }
         { literal-1    }    { literal-2    }
    GIVING { identifier-3 [ ROUNDED ] } . . .
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-MULTIPLY ]
```

2.35 GENERAL FORMAT FOR OPEN STATEMENT

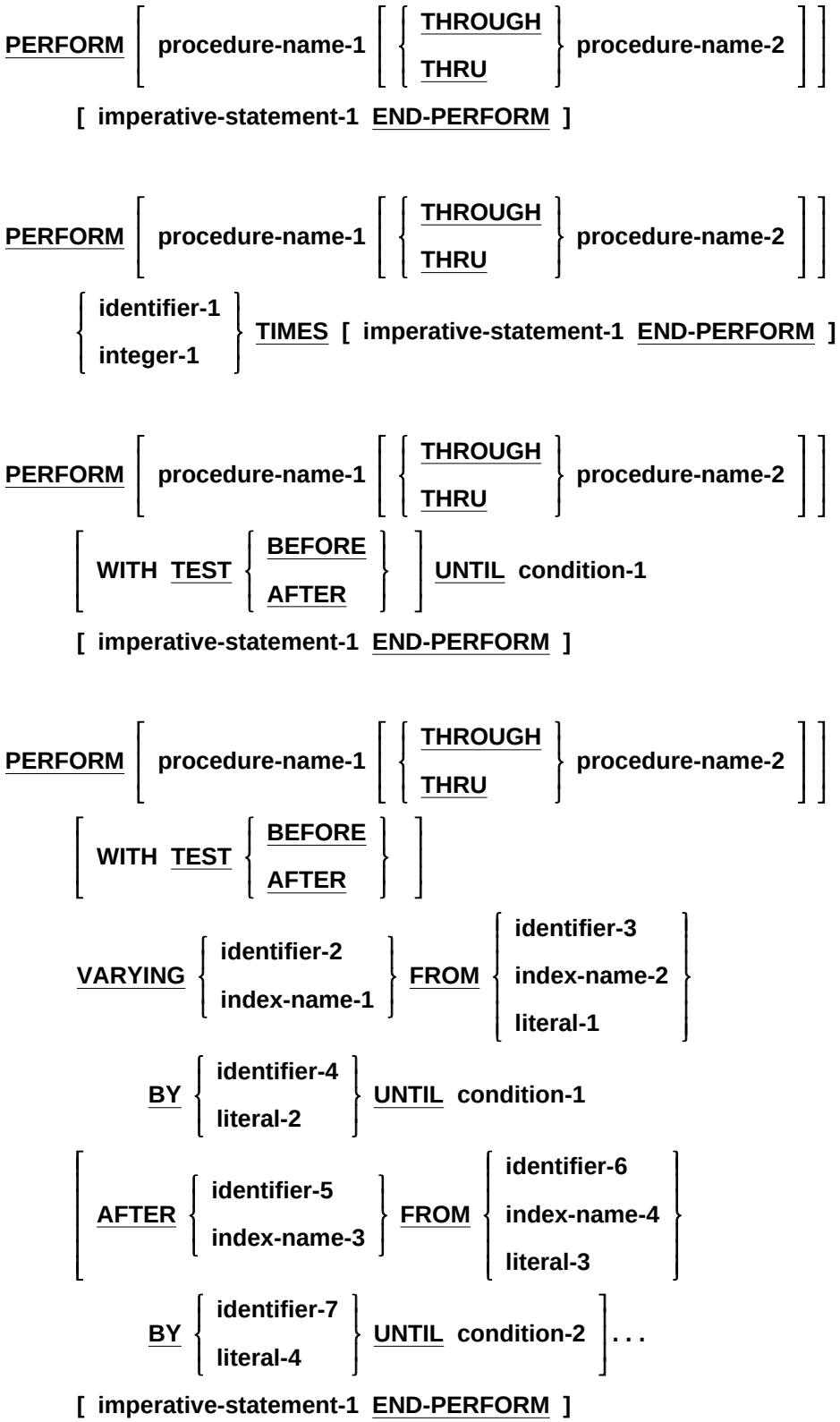
2.35.1 Sequential or Line Sequential File



2.35.2 Relative or Indexed File



2.36 GENERAL FORMAT FOR PERFORM STATEMENT



2.37 GENERAL FORMAT FOR READ STATEMENT

2.37.1 Sequential, Line Sequential, Relative or Indexed File

```
READ file-name-1 [ NEXT ] RECORD [ INTO identifier-1 ]
     [ WITH [ NO ] LOCK ]                                    E
     [ AT END imperative-statement-1 ]
     [ NOT AT END imperative-statement-2 ]
     [ END-READ ]
```

2.37.2 Relative File

```
READ file-name-1 RECORD [ INTO identifier-1 ]
     [ WITH [ NO ] LOCK ]                                    E
     [ INVALID KEY imperative-statement-1 ]
     [ NOT INVALID KEY imperative-statement-2 ]
     [ END-READ ]
```

2.37.3 Indexed File

```
READ file-name-1 RECORD [ INTO identifier-1 ]
     [ WITH [ NO ] LOCK ]                                    E
     [ KEY IS data-name-1 ]
     [ INVALID KEY imperative-statement-1 ]
     [ NOT INVALID KEY imperative-statement-2 ]
     [ END-READ ]
```

2.38 GENERAL FORMAT FOR RELEASE STATEMENT

RELEASE record-name-1 [FROM identifier-1]

2.39 GENERAL FORMAT FOR REPLACE STATEMENT

REPLACE { ==pseudo-text-1== BY ==pseudo-text-2== } . . .

REPLACE OFF

2.40 GENERAL FORMAT FOR RETURN STATEMENT

```
RETURN file-name-1 RECORD [ INTO identifier-1 ]
    AT END imperative-statement-1
    [ NOT AT END imperative-statement-2 ]
    [ END-RETURN ]
```

2.41 GENERAL FORMAT FOR REWRITE STATEMENT

2.41.1 Sequential File

```
REWRITE record-name-1 [ FROM identifier-1 ]
        [ END-REWRITE ]
```

2.41.2 Relative or Indexed File

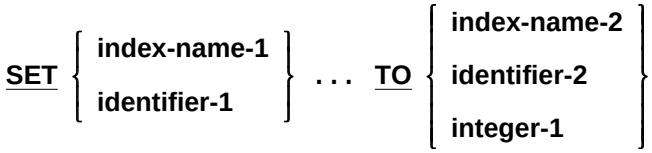
```
REWRITE record-name-1 [ FROM identifier-1 ]
        [ INVALID KEY imperative-statement-1 ]
        [ NOT INVALID KEY imperative-statement-2 ]
        [ END-REWRITE ]
```

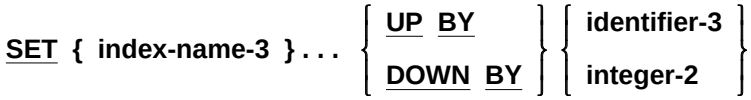
2.42 GENERAL FORMAT FOR SEARCH STATEMENT

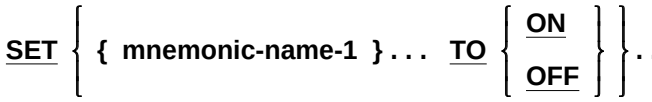
```
SEARCH identifier-1 [ VARYING { identifier-2   } ]
                              { index-name-1   }
    [ AT END imperative-statement-1 ]
    { WHEN condition-1 { imperative-statement-2 } } ...
                       { NEXT SENTENCE          }
    [ END-SEARCH ]
```

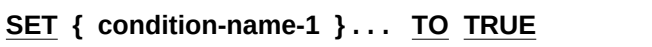
```
SEARCH ALL identifier-1 [ AT END imperative-statement-1 ]
    WHEN { data-name-1 { IS EQUAL TO } { identifier-3   } }
         {             { IS =       } { literal-1      } }
         {                           { arithmetic-expression-1 } }
         { condition-name-1                                    }
    [ AND { data-name-2 { IS EQUAL TO } { identifier-4   } } ] ...
          {             { IS =       } { literal-2      } }
          {                           { arithmetic-expression-2 } }
          { condition-name-2                                    }
    { imperative-statement-2 }
    { NEXT SENTENCE          }
    [ END-SEARCH ]
```

2.43 GENERAL FORMAT FOR SET STATEMENT

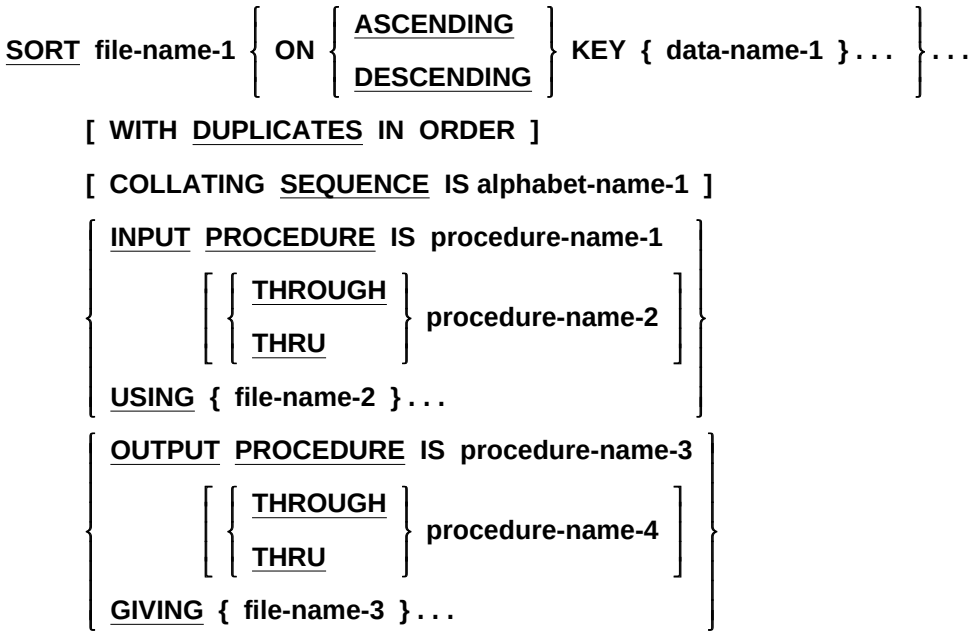




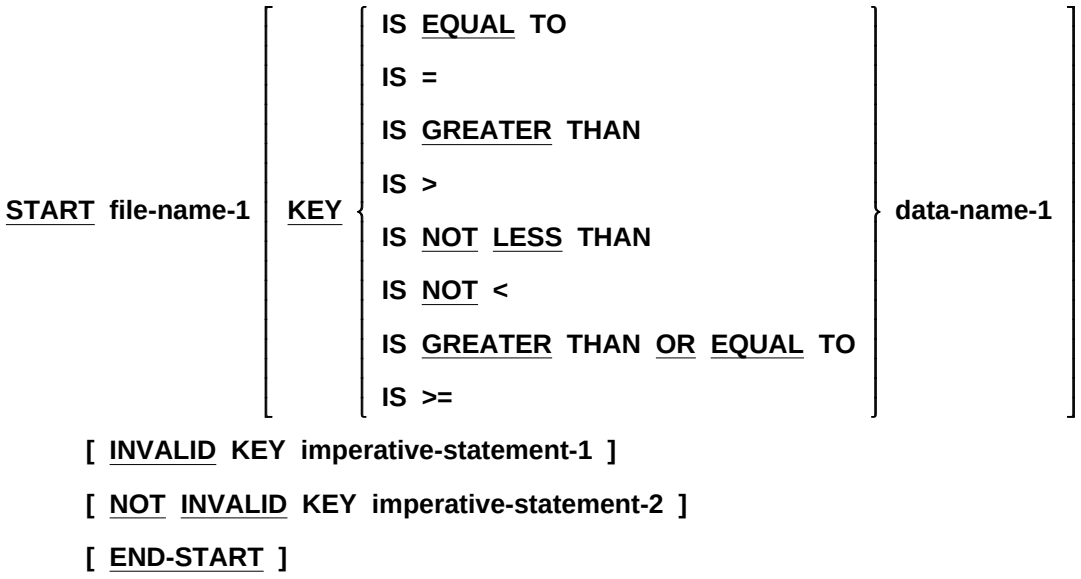




2.44 GENERAL FORMAT FOR SORT STATEMENT



2.45 GENERAL FORMAT FOR START STATEMENT



2.46 GENERAL FORMAT FOR STOP STATEMENT

STOP RUN

2.47 GENERAL FORMAT FOR STRING STATEMENT

INTO identifier-3

[WITH POINTER identifier-4]

[ON OVERFLOW imperative-statement-1]

[NOT ON OVERFLOW imperative-statement-2]

[END-STRING]

2.48 GENERAL FORMAT FOR SUBTRACT STATEMENT

```
SUBTRACT { identifier-1 }...  FROM { identifier-2 [ ROUNDED ] }...
         { literal-1    }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-SUBTRACT ]
```

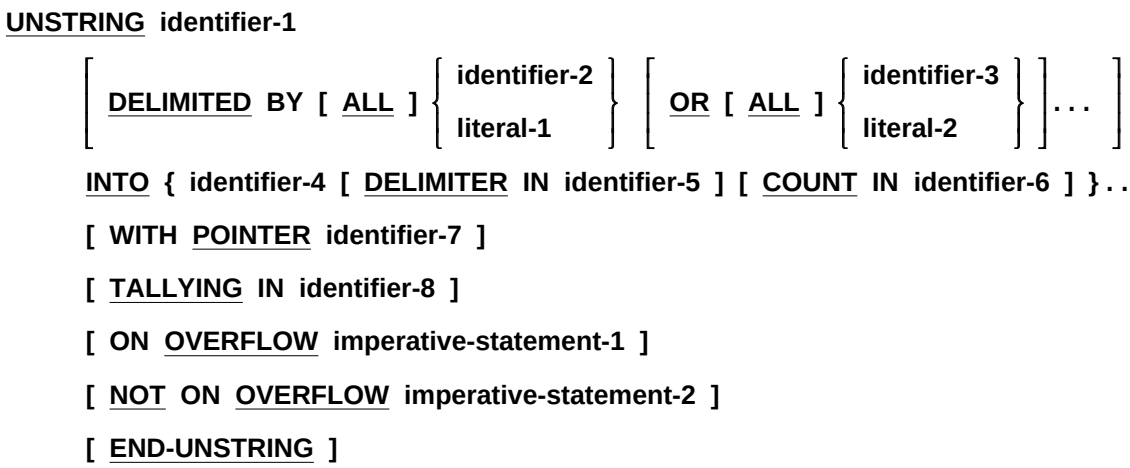
```
SUBTRACT { identifier-1 }...  FROM { identifier-2 }
         { literal-1    }          { literal-2    }
    GIVING { identifier-3 [ ROUNDED ] }...
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-SUBTRACT ]
```

```
SUBTRACT { CORRESPONDING } identifier-1 FROM identifier-2 [ ROUNDED ]
         { CORR          }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-SUBTRACT ]
```

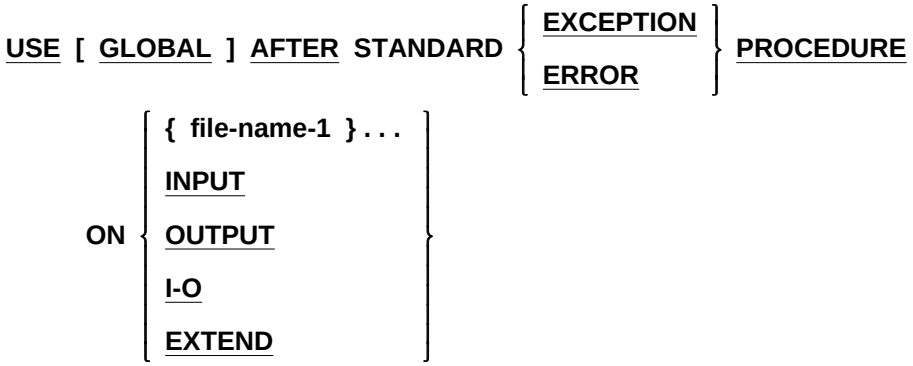
2.49 GENERAL FORMAT FOR UNLOCK STATEMENT

E

2.50 GENERAL FORMAT FOR UNSTRING STATEMENT



2.51 GENERAL FORMAT FOR USE STATEMENT



2.52 GENERAL FORMAT FOR WRITE STATEMENT

2.52.1 Sequential or Line Sequential File

```
WRITE record-name-1 [ FROM identifier-1 ]
    [ { BEFORE } ADVANCING { { integer-1    } [ LINE  ] } ]
    [ { AFTER  }           { { identifier-2 } [ LINES ] } ]
    [                      { PAGE                     } ]
    [ AT { END-OF-PAGE } imperative-statement-1 ]
    [ AT { EOP         }                        ]
    [ NOT AT { END-OF-PAGE } imperative-statement-2 ]
    [ NOT AT { EOP         }                        ]
    [ END-WRITE ]
```

2.52.2 Relative or Indexed File

```
WRITE record-name-1 [ FROM identifier-1 ]
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-WRITE ]
```

2.53 GENERAL FORMAT FOR CONDITIONS

2.53.1 Relation Condition

```
object-1 relational-operator-1 object-2
```

Object

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

Relational-Operator

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

2.53.2 Class Condition

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

2.53.3 Sign Condition

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

2.53.4 Condition-Name Condition

```
condition-name-1
```

2.53.5 Switch-Status Condition

```
condition-name-1
```

2.53.6 Negated Condition

```
NOT condition-1
```

2.53.7 Combined Condition

```
condition-1 { { AND } condition-2 }...
            { { OR  }             }
```

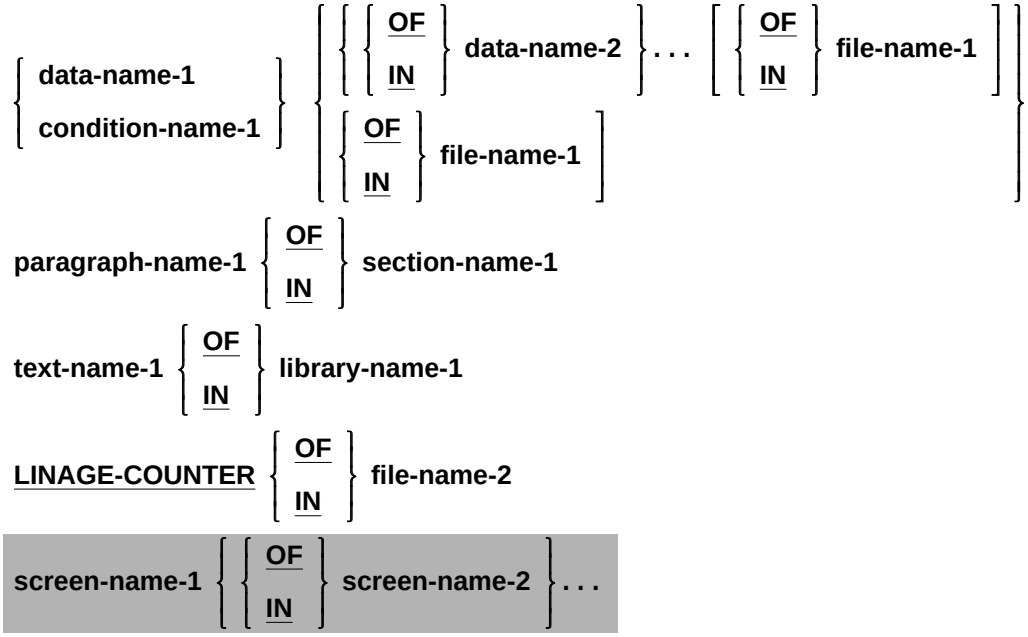
2.53.8 Abbreviated Combined Condition

```
relation-condition-1 { { AND } [ NOT ] [ relational-operator-1 ] object-1 }...
                     { { OR  }                                           }
```

2.54 GENERAL FORMAT FOR CONCATENATION-EXPRESSION

E

2.55 GENERAL FORMAT FOR QUALIFICATION



2.56 GENERAL FORMAT FOR IDENTIFIER

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

    [ subscript-1 ] [ reference-modifier-1 ]
```

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

2.56.1 Subscript

```
{ data-name-1     } ( { ALL                                          } ... )
{ condition-name-1 }   { integer-1                                    }
                       { data-name-2 [ { + } integer-2 ]              }
                       {             { - }             }
                       { index-name-1 [ { + } integer-3 ]             }
                       {              { - }             }
```

Note: *Data-name-1* and *condition-name-1* are shown in the above format to provide context and are not part of the subscript.

2.56.2 Reference-Modifier

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

    ( leftmost-character-position-1 : [ length-1 ] )
```

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

2.57 GENERAL FORMAT FOR INTRINSIC FUNCTIONS

FUNCTION ACOS (argument-1)

FUNCTION ANNUITY (argument-1 argument-2)

FUNCTION ASIN (argument-1)

FUNCTION ATAN (argument-1)

FUNCTION CHAR (argument-1)

FUNCTION COS (argument-1)

FUNCTION CURRENT-DATE

FUNCTION DATE-OF-INTEGER (argument-1)

FUNCTION DAY-OF-INTEGER (argument-1)

FUNCTION FACTORIAL (argument-1)

FUNCTION INTEGER (argument-1)

FUNCTION INTEGER-OF-DATE (argument-1)

FUNCTION INTEGER-OF-DAY (argument-1)

FUNCTION INTEGER-PART (argument-1)

FUNCTION LENGTH (argument-1)

FUNCTION LOG (argument-1)

FUNCTION LOG10 (argument-1)

FUNCTION LOWER-CASE (argument-1)

FUNCTION MAX ({ argument-1 } . . .)

FUNCTION MEAN ({ argument-1 } . . .)

FUNCTION MEDIAN ({ argument-1 } . . .)

FUNCTION MIDRANGE ({ argument-1 } . . .)

FUNCTION MIN ({ argument-1 } . . .)

FUNCTION MOD (argument-1 argument-2)

FUNCTION NUMVAL (argument-1)

FUNCTION NUMVAL-C (argument-1 [argument-2])

(continued overleaf)

General Format for Intrinsic Functions (continued)

```
FUNCTION ORD ( argument-1 )
FUNCTION ORD-MAX ( { argument-1 } . . . )
FUNCTION ORD-MIN ( { argument-1 } . . . )
FUNCTION PRESENT-VALUE ( argument-1 { argument-2 } . . . )
FUNCTION RANDOM [ ( argument-1 ) ]
FUNCTION RANGE ( { argument-1 } . . . )
FUNCTION REM ( argument-1 argument-2 )
FUNCTION REVERSE ( argument-1 )
FUNCTION SIN ( argument-1 )
FUNCTION SQRT ( argument-1 )
FUNCTION STANDARD-DEVIATION ( { argument-1 } . . . )
FUNCTION SUM ( { argument-1 } . . . )
FUNCTION TAN ( argument-1 )
FUNCTION UPPER-CASE ( argument-1 )
FUNCTION VARIANCE ( { argument-1 } . . . )
FUNCTION WHEN-COMPILED
```

See also **Section 7.1**, **Restrictions for Portable Programs** for a list of intrinsic functions that may cause portability problems.

Chapter 3

Definition of Extensions

3.1 INTRODUCTION

The specification of Standard COBOL is documented in ISO 1989:1985 (including ISO 1989/Amendment 1), so there is little value in repeating such detail in this specification. However, there are parts of X/Open COBOL where additional syntax or general rules describe details left to the implementor by ISO 1989:1985 and extensions to Standard COBOL. These details and rules are described here:

- Numeric Data Types and Representation
- Line Sequential I-O
- Special Register RETURN-CODE
- Passing Parameters by Value
- EBCDIC Character Code Set
- Concatenation of Nonnumeric and National-character Literals
- Free Form Reference Format

The following extensions are described in separate chapters:

- Screen Handling Module (**Chapter 4**)
- File Sharing and Record Locking (**Chapter 5**)
- Support for Internationalisation Facilities (**Chapter 6**, **Internationalisation**)
- Support for Multi-byte Character Sets (**Chapter 6**, **Internationalisation**)
- Access to Command Line Arguments and Environment Variables (**Chapter 8**, **COBOL in an X/Open Environment**)
- File (Re)assignment and Special Files (**Chapter 8**, **COBOL in an X/Open Environment**)
- Interface to C (**Chapter 8**, **COBOL in an X/Open Environment**)

The extensions introduce the following additional reserved words:

AUTO	CURSOR	LOWLIGHT
AUTOMATIC	END-ACCEPT	MANUAL
BACKGROUND-COLOR	END-DISPLAY	REQUIRED
BELL	EOL	RETURN-CODE
BLINK	EOS	REVERSE-VIDEO
COMP-3	ERASE	SCREEN
COMP-5	EXCLUSIVE	SECURE
COMPUTATIONAL-3	FOREGROUND-COLOR	UNDERLINE
COMPUTATIONAL-5	FULL	UNLOCK
CRT	HIGHLIGHT	

The extensions introduce the following additional character in the COBOL character set:

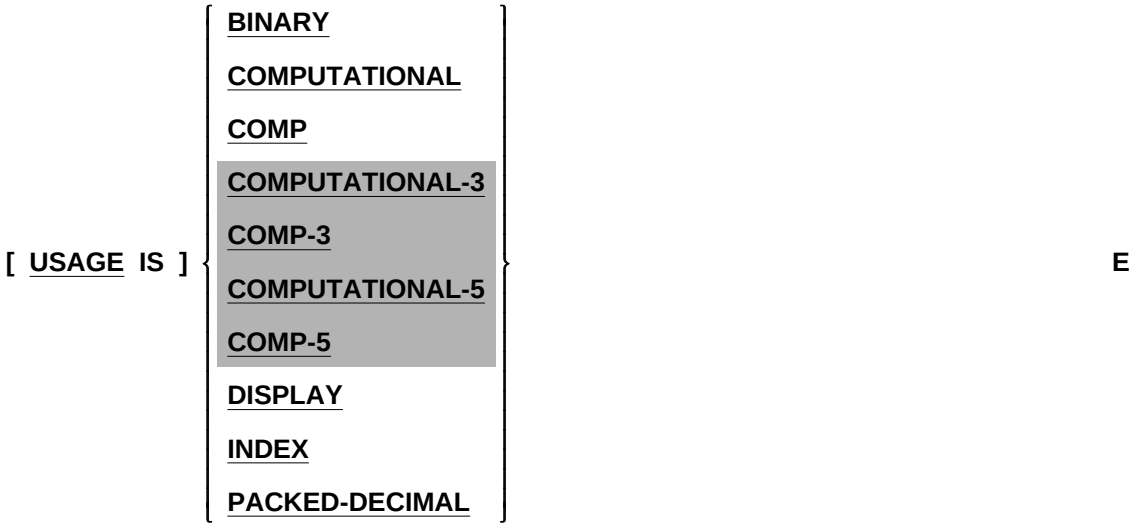
&

3.2 NUMERIC DATA TYPES AND REPRESENTATIONS

Function

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

General Format



Additional General Rules

1. Data items with level numbers 01 or 77 are aligned according to the requirements of the operating system. Elementary data items within records are aligned as described below.
2. USAGE DISPLAY

   The standard data format used to represent the data item is one ASCII byte for one character. The sign of signed numeric data described without the SEPARATE CHARACTER phrase in the SIGN clause is encoded by changing the leading or trailing digit optionally according to the SIGN clause. The encoding itself is implementation-dependent.
3. USAGE BINARY

   — Binary items are aligned on a byte boundary.

   — The representation is defined to be least significant digit on highest address.

   — For portable specifications the following table should be used:

PICTURE Specification	Length in Bytes
9(4), S9(4)	2
9(9), S9(9)	4
9(18), S9(18)	8

   — Negative numbers are coded as 2s complement.

4. USAGE COMPUTATIONAL, COMP

   The representation, the alignment, the size and the rules for the range of values the data item can hold are implementation-dependent.

5. USAGE COMPUTATIONAL-5, COMP-5

   — Typically, the representation is binary and negative numbers are coded as 2s complement. The representation matches that of C language integers.

   — The specification of PIC S9(4) is typically identical to the type **short int** in C.

   — The specification of PIC S9(9) is typically identical to the type **long int** in C.

   — Binary items are aligned on a byte boundary.

   — The size of the portable specification is the same as that defined for the BINARY data type.

   — For more details, refer to the X/Open C Language definition (see **Referenced Documents**).

6. USAGE PACKED-DECIMAL

   — The representation is two binary coded decimal digits per byte, ordered with the least significant digit in the highest address.

   — PACKED-DECIMAL data items are aligned on byte boundaries. When padding is required for an even number of digits, the leftmost byte shall have a zero as most significant digit.

   — The rightmost half byte shall indicate the sign.

      — x'F' is used for unsigned data items.

      — x'C' is used to indicate a positive value.

      — x'D' is used to indicate a negative value.

7. USAGE COMPUTATIONAL-3, COMP-3

   The same rules apply as for USAGE PACKED-DECIMAL.

8. USAGE INDEX

   The representation and the physical size of index data items are implementation-dependent.

Application Usage

Application writers should use data items specifying USAGE COMP-5 for inter-program communication between C programs and COBOL programs.

In the following cases, the USAGE clause specifying COMP, COMP-5 or INDEX should not be specified, and any signed numeric data should be described with the SIGN IS SEPARATE clause:

- for a data description entry, or any entry subordinate to it, that contains a REDEFINES clause

- in the data description entry, or any entry subordinate to the data description entry, referred to by *data-name-2* of the REDEFINES clause
- in the data description entries, or any entry subordinate to the data description entries, referred to by *data-name-2* through *data-name-3* of the RENAMES clause
- in the File Section

3.3 LINE SEQUENTIAL I-O

Function

The ORGANIZATION IS LINE SEQUENTIAL clause specifies a variant of sequential files compatible with the system text editor.

General Format

[ORGANIZATION IS] LINE SEQUENTIAL **E**

General Rules

All rules in Standard COBOL for sequential I-O apply in the same way for line sequential files with the following exceptions and extensions:

1. Each record contains a sequence of characters ending with a record terminator. The terminator is not counted in the length of the record.
2. Upon writing it is unspecified whether trailing blanks are removed prior to adding the record terminator. The characters in the record area from the first one up to and including the added record terminator constitute one record and are written to the file.
3. Upon reading the record, characters are read one at a time into the record area until:
   — the first record terminator is encountered. The record terminator is discarded and the remainder of the record area is filled with spaces.
   — the entire record area is filled with characters. If the first unread character is the record terminator it is discarded, otherwise the first unread character shall become the first character to be read by the next READ statement.
4. A line sequential file should only contain printable characters and the record terminator.
5. OPEN statement: for line sequential files only the open modes input, output or extend are possible. The open mode I-O is not possible.
6. WRITE statement: for line sequential files the ADVANCING phrase should not be used.

3.4 PASSING PARAMETERS BY VALUE

Function

The CALL statement causes control to be transferred from one object program to another within the run unit.

General Formats

```
CALL { identifier-1 }
     { literal-1   }
     [ USING { [ BY REFERENCE ] { identifier-2 } ... }
             { BY CONTENT { identifier-2 } ...       } ...
             { BY VALUE { identifier-3 } ...         }      ]   E
     [ ON OVERFLOW imperative-statement-1 ]
     [ END-CALL ]

CALL { identifier-1 }
     { literal-1   }
     [ USING { [ BY REFERENCE ] { identifier-2 } ... }
             { BY CONTENT { identifier-2 } ...       } ...
             { BY VALUE { identifier-3 } ...         }      ]   E
     [ ON EXCEPTION imperative-statement-1 ]
     [ NOT ON EXCEPTION imperative-statement-2 ]
     [ END-CALL ]
```

General Rules

All rules in Standard COBOL for the CALL statement are applicable with the following extensions:

1. The BY CONTENT, BY REFERENCE and BY VALUE phrases are transitive across the parameters which follow them until another BY CONTENT, BY REFERENCE or BY VALUE phrase is encountered. If no BY CONTENT, BY REFERENCE or BY VALUE phrase is specified for the first parameter, the BY REFERENCE phrase is assumed.
2. If the BY VALUE phrase is specified or implied for a parameter, the C language conventions for passing a parameter (by value) are used and only the value of the parameter is passed to the called program. The called program can access this value and modify it, but the value of the original parameter as specified in the calling program is not modified.

3. If the BY VALUE phrase is specified or implied for any parameter, the effect of passing that parameter is undefined except when:

   — calling a C program or a program written in another language that accepts value parameters

   — defining *identifier-3* as a 2 or 4-byte data item with USAGE COMP-5 or a 1-byte data item

3.5 SPECIAL REGISTER RETURN-CODE

A special register RETURN-CODE implicitly defined with GLOBAL scope is capable of holding a signed integer value less than 100000. This special register may be used to set a function return value for a calling C program or the calling shell, or to retrieve the value returned from a called program. For more details see **Chapter 8**, **COBOL in an X/Open Environment**.

3.6 EBCDIC CHARACTER CODE SET

The EBCDIC character code set, collating sequence and relation to STANDARD-1, is given in the following table. The mapping of character codes that are not mentioned is implementation-dependent. The EBCDIC codes 4A, 5A, 5B, 7B, 7C and E0 may translate to different STANDARD-1 codes in other countries.

US char	EBCDIC code	STANDARD-1 code
NUL	00	00
SOH	01	01
STX	02	02
ETX	03	03
HT	05	09
DEL	07	7F
VT	0B	0B
FF	0C	0C
CR	0D	0D
SO	0E	0E
SI	0F	0F
DLE	10	10
BS	16	08
CAN	18	18
EM	19	19
IFS	1C	1C
IGS	1D	1D
IRS	1E	1E
IUS	1F	1F
LF	25	0A
ETB	26	17
ESC	27	1B
ENQ	2D	05
ACK	2E	06
BEL	2F	07
SYN	32	16
EOT	37	04
NAK	3D	15
SUB	3F	1A
space	40	20
¢ [	4A	5B
.	4B	2E
<	4C	3C
(	4D	28
+	4E	2B
| !	4F	21
&	50	26
!]	5A	5D
$	5B	24
*	5C	2A
)	5D	29
;	5E	3B
¬ ^	5F	5E

US char	EBCDIC code	STANDARD-1 code
-	60	2D
/	61	2F
|	6A	7C
,	6B	2C
%	6C	25
_	6D	5F
>	6E	3E
?	6F	3F
‘	79	60
:	7A	3A
#	7B	23
@	7C	40
’	7D	27
=	7E	3D
"	7F	22
a	81	61
	:	
i	89	69
j	91	6A
	:	
r	99	72
~	A1	7E
s	A2	73
	:	
z	A9	7A
{	C0	7B
A	C1	41
	:	
I	C9	49
}	D0	7D
J	D1	4A
	:	
R	D9	52
\	E0	5C
S	E2	53
	:	
Z	E9	5A
0	F0	30
	:	
9	F9	39

3.7 CONCATENATION EXPRESSIONS

Function

The concatenation expression concatenates literals.

General Format

Syntax Rules

1. *Literal-1* and *literal-2* must both be a nonnumeric literal, or they must both be a national-character literal.
2. *Literal-1* and *literal-2* cannot be figurative constants that begin with the word ALL.

General Rules

1. The value of a concatenation expression is the concatenation of the literals and figurative constants it comprises. The separator quotation mark that delimits the nonnumeric literals, and the separator space that precedes and follows the concatenation operator, are not part of the value of the concatenation expression.
2. A concatenation expression is exactly equivalent to a nonnumeric or national-character literal with the same value, and can be used anywhere that nonnumeric or national-character literal can be used.
3. A concatenation expression can be preceded by the word ALL to form a figurative constant.
4. The class of the concatenation expression resulting from the concatenation operation is:
   - — the same class as the operands
   - — when one of the operands is a figurative constant, the class of the literal or concatenation expression that constitutes the other operand
   - — when both the operands are figurative constants, the class alphanumeric
5. Comment lines and blank lines may be interspersed between the operands and operators of a concatenation expression.

3.8 FREE FORM REFERENCE FORMAT

To facilitate the use of multi-byte data in COBOL source programs, an alternate reference format is introduced in this X/Open definition. The alternative reference format to fixed form is free form. The X/Open COBOL definition for free form reference format is modelled after the CODASYL COBOL reference format for free form source. The following section defines only the free form reference format.

3.8.1 General Description

The X/Open COBOL free form reference format, which provides a standard method for describing COBOL source programs and COBOL library text, is described in terms of character positions in a line.

The X/Open COBOL free form reference format defines a line as a logical record containing a maximum of 80 character positions. The position of a character is defined by the number of characters preceding the character on a line, discarding any control information that may be present. The source lines may vary in length due to the number of bytes that are needed to represent the maximum of 80 character positions in the supported character set.

Each division of the COBOL program must be written according to the rules for this reference format. The text of the source program may appear anywhere on a line with the following exceptions:

- When appearing in the first character position of a line, the single-byte symbols ''*'' and ''/'' are always interpreted as indicating comment lines.
- When appearing in the first two character positions of a line, the single-byte symbol pair ''D '' (D followed by space) is always interpreted as a debugging line.

3.8.2 Continuation of Lines

Any entry, phrase, pseudo-text or sentence consisting of more than one character-string may be continued by writing some of the character-strings and separators that constitute it on successive lines. These subsequent lines are called the continuation lines. The line being continued is called the continued line. The last non-blank character of every line is always treated as if it were followed by a space. That is, COBOL words, numeric literals and PICTURE character-strings may not be continued across continuation lines. Nonnumeric literals may be continued across continuation lines using the concatenation operator.

3.8.3 Comment Lines

A comment line is any line with a single-byte ''*'' or ''/'' in the first character position of the line. A comment line can appear as any line in a source program after the Identification Division header, and as any line in library text of a COBOL library. A ''/'' in the first character position of the line causes page ejection prior to printing the comment line in the listing of the source program. A ''*'' in the first character position of the line causes printing of the line at the next available line position in the listing.

A comment may be composed of any combination of single-byte and multi-byte characters following its beginning indicator (''*'' or ''/''). A comment, including its beginning indicator (''*'' or ''/''), is copied to the listing. A comment receives no syntax

checking; it serves only as documentation.

3.8.4 Debugging Lines

A debugging line is any line with a single-byte upper-case or lower-case ‘‘D’’ in the first character of the line immediately followed by a single-byte space. Multiple, successive debugging lines are permitted, but each must contain ‘‘D ’’ in their first two bytes.

A debugging line is permitted only in a separately compiled program after the OBJECT-COMPUTER paragraph.

The contents of a debugging line must be such that a syntactically correct program is formed with or without the debugging lines being considered as comment lines.

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

3.8.5 Restrictions

In a compilation unit, either fixed form or free form reference format is selected, and the selection method is implementation-dependent, but shall not be selected as a result of any construct contained in the program.

Fixed form and free form reference format should not be combined in one compilation unit. This includes the case when text is copied from a COBOL library.

See **Appendix A**, **Future Directions** for removal of these restrictions.

Chapter 4

Screen Handling Module

4.1 INTRODUCTION

The Screen Handling module provides the user with enhanced screen handling facilities. It consists of a Screen Section, additional formats of ACCEPT and DISPLAY statements, and additions to the SPECIAL-NAMES paragraph of the Environment Division.

4.2 ENVIRONMENT DIVISION

4.2.1 The SPECIAL-NAMES Paragraph

Function

The SPECIAL-NAMES paragraph provides a means for relating implementor-names to user specified mnemonic-names.

General Format

```
SPECIAL-NAMES .
     [ CURSOR IS data-name-1 ]
     [ CRT STATUS IS data-name-2 ] .
```

E

Syntax Rules

1. *Data-name-1* shall be declared in the Working-Storage Section of the program, and shall be either 4 or 6 characters in length.
2. *Data-name-1* shall describe either an elementary unsigned numeric integer, described implicitly or explicitly as USAGE IS DISPLAY, or shall describe a group item consisting of 2 such elementary data items.
3. *Data-name-2* shall be declared in the Working-Storage Section of the program and shall be a group item 3 characters in length.

General Rules

1. *Data-name-1* specifies the cursor address as used by the ACCEPT statement. If CURSOR is not specified the default cursor position on executing an ACCEPT statement is the start of the first input or update field of the screen.
2. The CURSOR clause enables a program to retain the position of the cursor at the end of the execution of the ACCEPT statement, or to specify its initial position at the start of an ACCEPT statement.
3. If *data-name-1* is four characters in length, the first two characters are interpreted as line number, and the second two as column number. If *data-name-1* is six characters in length, the first three characters are interpreted as line number, and the second three as column number.
4. If the cursor address does not indicate a position within an input or update field, the initial position of the cursor is undefined. If the initial position of the cursor is undefined, it is implementation-dependent whether its position at the end of the ACCEPT statement is stored in *data-name-1*.
5. If the CRT STATUS clause is specified, *data-name-2* shall be updated after every ACCEPT statement.
6. The first character of *data-name-2* is CRT Status Key 1 and should be described as PICTURE 9 USAGE DISPLAY. It indicates the condition that caused the termination of

the accept operation. The possible values are:

‘0’ indicates a terminator key or auto-skip out of the final field

‘1’ indicates a user-defined function key

‘2’ indicates an implementation-dependent function key

‘9’ indicates an error

7. A termination that returns a value of ‘0’ is a normal termination.

8. The second character of *data-name-2* is CRT Status Key 2 and should be described as PICTURE X. It contains a code giving further details of the condition that terminated the accept operation. Its possible values depend on the value in CRT Status Key 1 as shown in the following table.

KEY 1	KEY 2	MEANING
‘0’	‘0’	The operator pressed a terminator key.
‘0’	‘1’	Auto-skip out of the last field.
‘1’	x‘00’-x‘FF’	The function key number.
‘2’	x‘00’-x‘FF’	The function key number.
‘9’	x‘00’	No items fall within the screen.

9. The third character of *data-name-2* is not currently defined.

4.3 DATA DIVISION

4.3.1 The Screen Section

Function

The Screen Section provides screen handling facilities for use with ACCEPT and DISPLAY statements.

General Format

```
SCREEN SECTION .
[ screen-description-entry ] . . .
```

E

Syntax Rule

The Screen Section shall follow all sections of the Data Division defined in Standard COBOL.

4.3.2 The Screen Description Entry

Function

A screen description entry specifies the attributes, behaviour, size and location for a referenced screen item.

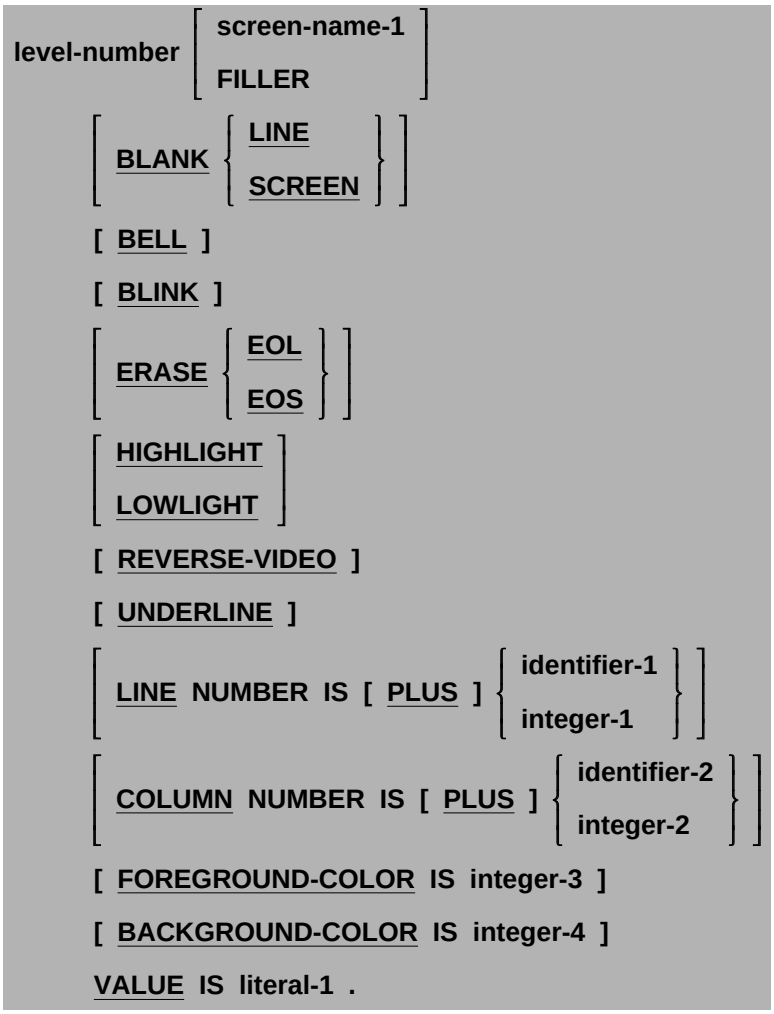
General Format

Format 1

```
level-number [ screen-name-1 ]
             [ FILLER        ]
      [ BLANK SCREEN ]
      [ FOREGROUND-COLOR IS integer-3 ]
      [ BACKGROUND-COLOR IS integer-4 ]
      [ AUTO ]
      [ SECURE ]
      [ REQUIRED ]
      [ [ USAGE IS ] DISPLAY ]
      [ [ SIGN IS ] { LEADING  } [ SEPARATE CHARACTER ] ]
                    { TRAILING }
      [ FULL ] .
```

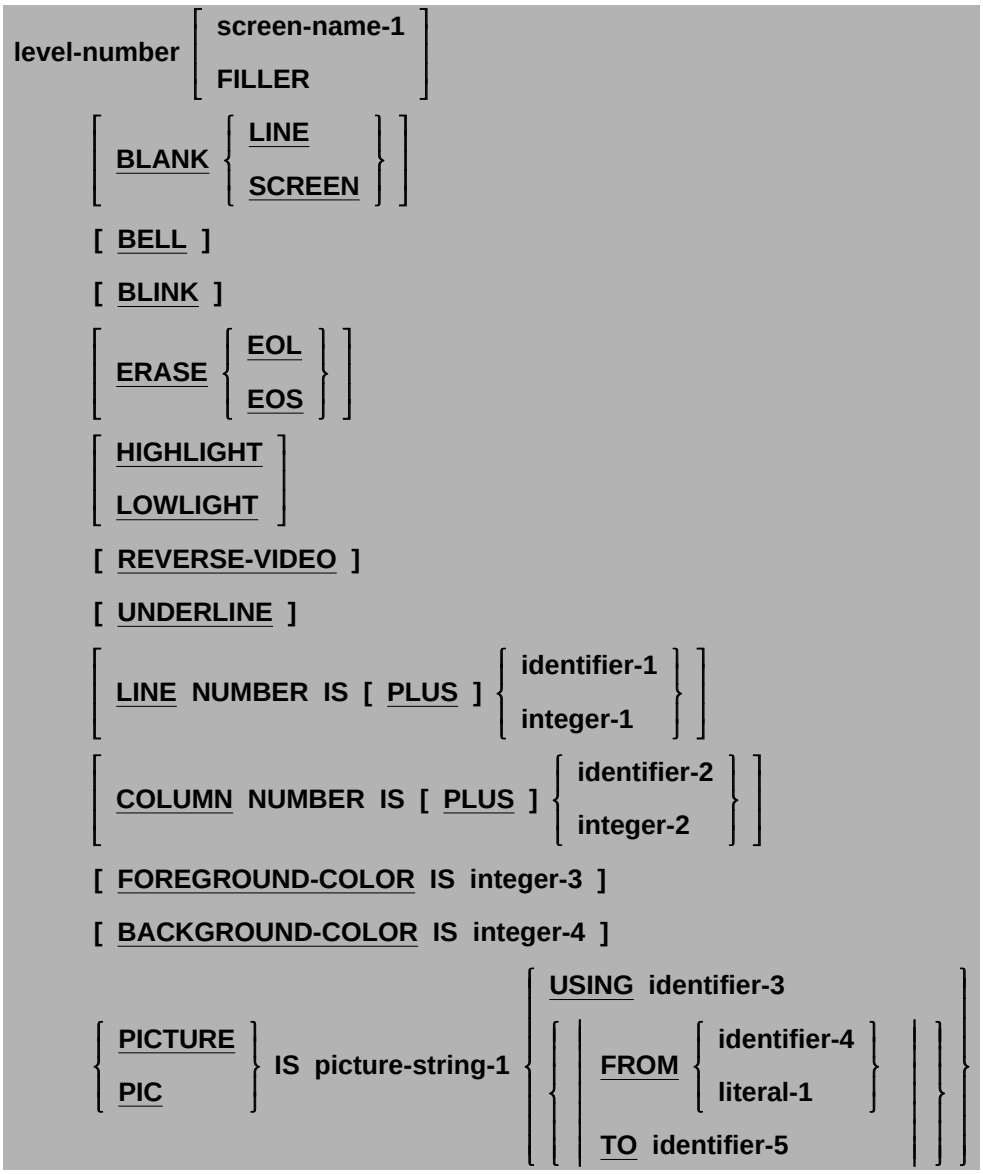
E

Format 2


level-number
screen-name-1
FILLER
BLANK
LINE
SCREEN
BELL
BLINK
ERASE
EOL
EOS
HIGHLIGHT
LOWLIGHT
REVERSE-VIDEO
UNDERLINE
LINE NUMBER IS PLUS
identifier-1
integer-1
COLUMN NUMBER IS PLUS
identifier-2
integer-2
FOREGROUND-COLOR IS integer-3
BACKGROUND-COLOR IS integer-4
VALUE IS literal-1 .

E

Format 3



E

(continued overleaf)

Format 3 (continued)

```
[ [ USAGE IS ] DISPLAY ]
[ BLANK WHEN ZERO ]
[ { JUSTIFIED } RIGHT ]
[ { JUST      }       ]
[ [ SIGN IS ] { LEADING  } [ SEPARATE CHARACTER ] ]
              { TRAILING }                                  E
[ AUTO ]
[ SECURE ]
[ REQUIRED ]
[ FULL ] .
```

Syntax Rules

1. Each screen description entry shall start with a level number from 01 through 49.
2. Each elementary item shall contain at least one of the following clauses: BELL, BLANK, COLUMN, LINE, PICTURE or VALUE.
3. If the same clause is specified more than once for the same screen item, the clause which appears at the lowest level within the hierarchy is the one which takes effect.
4. If either *screen-name-1* or the key word FILLER is specified, it shall be the first word following the level number in each screen description entry. *Screen-name-1* assigns a name to the screen item described in the screen description and shall conform to the rules for user-defined names.
5. If the FILLER or *screen-name-1* clause is omitted, the screen item being described is treated as though FILLER had been specified.
6. The key word FILLER may be used to name a screen item. Under no circumstances can a FILLER item be referred to explicitly.
7. Each level 01 item shall have a screen-name.
8. A screen item can only be referenced in an ACCEPT or DISPLAY statement, as described later in this chapter.
9. The data items in the FROM, TO and USING phrases are associated with the screen item. The USING phrase is equivalent to the combination of FROM and TO phrases, each specifying the same identifier.
10. An ACCEPT statement may refer to a group screen item containing screen items with FROM or VALUE clauses only if the group also contains screen items with TO or USING clauses.

General Rules

1. Format 1 is used for group descriptions.
2. Format 2 is used for literal screen elements.
3. Format 3 is used for data item screen elements.
   - An input field is a screen item whose description contains a TO phrase.
   - An output field is a screen item whose description contains a FROM phrase.
   - An update field is a screen item whose description contains a USING phrase.
4. The rules for update fields also apply to screen items whose description contains both a FROM and a TO phrase that may or may not reference the same identifier.
5. If the SECURE clause is specified in format 1, it applies to each input field in the group.
6. If the AUTO, BACKGROUND-COLOR, FOREGROUND-COLOR, FULL or REQUIRED clause is specified in format 1, it applies to each input and update field in the group.
7. LINE and COLUMN clauses should not be specified within a screen record description entry in such a way that fields overlap on the screen or fall outside the COLS or LINES limits defined for the terminal concerned.

4.3.3 The AUTO Clause

Function

The AUTO clause causes the cursor to be automatically moved to the next field when, during execution of an ACCEPT statement, the last character of the input or update field, whose definition contains this clause, is entered.

General Format

AUTO **E**

Syntax Rules

1. The AUTO clause may not be specified in a literal screen element entry.

General Rules

1. If the AUTO clause is specified at group level, it applies to each input and update item in that group.
2. The AUTO clause only has an effect during the execution of an ACCEPT statement. It is ignored during the execution of a DISPLAY statement.
3. The AUTO clause is ignored for an elementary output field.
4. If there is only one field to be input, or if the field is the last one of the screen, the ACCEPT statement is completed.

4.3.4 The BACKGROUND-COLOR Clause

Function

The BACKGROUND-COLOR clause specifies the background colour for the screen item.

General Format

BACKGROUND-COLOR IS integer-4 **E**

Syntax Rules

1. The clause is allowed with any screen item.
2. If the clause is specified at group level, it applies to all elementary subordinate items.
3. *Integer-4* must be a value from 0 to 7.

General Rules

1. This clause is effective only with colour screens.
2. *Integer-4* specifies the background colour for the screen item. The colours and their corresponding values are shown below:

   0 black

   1 blue

   2 green

   3 cyan

   4 red

   5 magenta

   6 brown/yellow

   7 white
3. If the clause is not specified, the background colour is black.

4.3.5 The BELL Clause

Function

The BELL clause causes the workstation audio tone to sound.

General Format

BELL **E**

Syntax Rule

The BELL clause may be specified only for elementary screen description entries.

General Rule

The use of this clause results in the audio tone sounding when the screen data element in which it is specified is processed during the execution of a DISPLAY statement. The audio tone sounds at the start of the display of the element.

4.3.6 The BLANK Clause

Function

The BLANK clause clears a screen line or clears the whole screen before displaying.

General Format

E

Syntax Rules

1. The BLANK SCREEN clause may be specified in any screen description entry.
2. The BLANK LINE clause may be specified only for elementary entries.

General Rules

1. The BLANK SCREEN clause is executed before DISPLAY of a screen data item, no matter where it appears. If the clause is specified, the screen is cleared and the cursor is placed at line 1, column 1.
2. When BLANK LINE is specified, blanking begins at column 1 of the line specified for the screen data element in whose description it is included, and continues through to the end of the line.
3. If neither the BLANK nor the ERASE clause is specified, only the particular character positions corresponding to the screen data element are modified when the element is displayed. The rest of the screen content remains the same.
4. The BLANK SCREEN clause in combination with the BACKGROUND-COLOR clause for the same screen item, or for one that it is subordinate to, establishes the default background colour to be used until the same combination is encountered specifying another background colour (whether in the same DISPLAY statement or another).
5. The BLANK SCREEN clause in combination with the FOREGROUND-COLOR clause for the same screen item, or for one that it is subordinate to, establishes the default foreground colour to be used until the same combination is encountered specifying another foreground colour (whether in the same DISPLAY statement or another).
6. The BLANK clause is ignored in an ACCEPT statement.

4.3.7 The BLANK WHEN ZERO Clause

Function

The BLANK WHEN ZERO clause displays blanks (space characters) when the value of a data item is zero.

General Format

BLANK WHEN ZERO **E**

General Rule

The BLANK WHEN ZERO clause is ignored for an elementary input field.

For further rules refer to the BLANK WHEN ZERO clause in ISO 1989:1985, The Nucleus Module.

4.3.8 The BLINK Clause

Function

The BLINK clause blinks an item when it is displayed on the screen.

General Format

BLINK **E**

Syntax Rule

The BLINK clause may be specified only in an elementary screen description entry.

4.3.9 The COLUMN Clause

Function

The COLUMN clause specifies the horizontal screen coordinate for a screen data element.

General Format

```
COLUMN NUMBER IS [ PLUS ] { identifier-2 }                    E
                          { integer-2    }
```

Syntax Rules

1. The COLUMN clause can be specified only for an elementary item.
2. *Integer-2* shall be unsigned and *identifier-2* shall be described as an elementary unsigned numeric integer data item. *Identifier-2* may not be subscripted.

General Rules

1. The COLUMN clause specifies the column in which the screen item is to appear on the screen in an accept or display operation.
2. If the COLUMN clause has an identifier or an integer but does not specify PLUS, the clause gives an absolute column number. Column 1 is the column number specified in the AT phrase of the DISPLAY or ACCEPT statement. If the AT phrase is not specified within the statement, then column 1 is the first column of the screen.
3. If PLUS is specified in the COLUMN clause, then the column number is relative to that at which the preceding item ends, regardless of whether or not the statement displays the preceding item on the screen. This depends on the current length of the item derived from the PICTURE or VALUE clause. The counting of column numbers restarts at a level 01 item at column 1.
4. The first screen item of a screen referenced in an ACCEPT or DISPLAY statement must not specify a relative column number.
5. A setting of COLUMN 1 is assumed for screen descriptions which specify the LINE clause but omit the COLUMN clause.
6. If both the LINE clause and the COLUMN clause are omitted, the following apply:
   a. If no previous screen data element has been defined, LINE 1 COLUMN 1 of the screen is assumed.
   b. If a previous screen data element has been defined, the line of that previous element and COLUMN PLUS 1 is assumed. The screen data element then starts immediately following the preceding screen data element.

4.3.10 The ERASE Clause

Function

The ERASE clause clears part of the line or the screen starting at the cursor position.

General Format

E

Syntax Rule

The ERASE clause may be specified only for elementary entries.

General Rules

1. When ERASE EOL is specified, blanking begins at the line and column coordinates specified for the screen data element in whose description it is included, and continues to the end of the line.
2. When ERASE EOS is specified, blanking begins at the line and column coordinates specified for the screen data element in whose description it is included, and continues through to the end of the screen.
3. If neither the BLANK nor the ERASE clause is specified, only the particular character positions corresponding to the screen data element are modified when the element is displayed. The rest of the screen content remains the same.
4. The clause is ignored in an ACCEPT statement.

4.3.11 The FOREGROUND-COLOR Clause

Function

The FOREGROUND-COLOR clause specifies the foreground colour for the screen item.

General Format

FOREGROUND-COLOR IS integer-3 **E**

Syntax Rules

1. The FOREGROUND-COLOR clause is allowed with any screen item.
2. If the clause is specified at group level, it applies to all elementary subordinate items.
3. *Integer-3* must be a value from 0 to 7.

General Rules

1. This clause is effective only with colour screens.
2. *Integer-3* specifies the foreground colour for the screen item. The colours and their corresponding values are shown below.

   0 black

   1 blue

   2 green

   3 cyan

   4 red

   5 magenta

   6 brown/yellow

   7 white
3. If the clause is not specified, the foreground colour is white.
4. If the HIGHLIGHT clause is also specified, foreground and background colours are brightened and lightened; for example, black may become grey.

4.3.12 The FULL Clause

Function

The FULL clause specifies that the operator must either leave the screen item completely empty or fill it entirely with data.

General Format

FULL **E**

Syntax Rule

1. If the FULL clause is specified for a screen description entry, the JUSTIFIED clause may not be specified.
2. The FULL clause is valid only in input or update fields.
3. If the clause is specified at group level, it applies to all subordinate input or update fields.

General Rules

1. The FULL clause is effective during the execution of any ACCEPT statement that causes the screen item to be accepted, provided the cursor enters the screen item at some time during the accept operation. Until this clause is satisfied, terminator keystrokes are rejected.
2. If the screen item is alphanumeric, then to satisfy this clause either the entire item must contain all spaces, or both the first and last character positions must contain non-space characters.
3. If the screen item is numeric or numeric edited, then to satisfy the clause either the value must be zero or there must be no digit position in which zero suppression has taken effect.
4. For update fields, the FULL clause can be satisfied by the contents of the identifier, or literal referenced in the FROM or USING phrase of the PICTURE clause, as well as operator keyed data.
5. The FULL clause may not be effective if a function key is used to terminate the accept operation.
6. The specification of the FULL and REQUIRED clauses together requires that the user must always entirely fill the field.
7. The FULL clause is ignored for an elementary output field.

4.3.13 The HIGHLIGHT Clause

Function

The HIGHLIGHT clause specifies that the field is to appear on the screen with the highest intensity.

General Format

HIGHLIGHT **E**

Syntax Rule

The HIGHLIGHT clause may be specified only for an elementary screen description entry.

4.3.14 The JUSTIFIED Clause

Function

The JUSTIFIED clause specifies non-standard positioning of data within a screen data element.

General Format

E

The rules for the JUSTIFIED clause are the same as in ISO 1989:1985, The Nucleus Module.

4.3.15 The LINE Clause

Function

The LINE clause specifies the vertical screen coordinate for a screen data element.

General Format

E

Syntax Rules

1. The LINE clause may be specified only for an elementary item.
2. *Integer-1* shall be unsigned and *identifier-1* shall be described as an elementary unsigned numeric integer data item. *Identifier-1* may not be subscripted.

General Rules

1. The LINE clause, in conjunction with the COLUMN clause, establishes the starting coordinates for a screen data element. The LINE clause specifies the vertical coordinate.
2. The LINE clause without the PLUS phrase specifies the absolute screen line position of the screen data element.
3. Line 1 is the line number specified in the AT phrase of the ACCEPT or DISPLAY statement. If the AT phrase is not specified within the statement, then line 1 is the first line of the screen.
4. The LINE clause with the PLUS phrase specifies a screen line number relative to the line containing the last defined screen data element.
5. The first screen item of a screen referenced in an ACCEPT or DISPLAY statement must not specify a relative line number.
6. If the LINE clause is omitted, the following apply:
    a. If no previous screen data element has been defined, LINE 1 of the screen is assumed.
    b. If a previous screen data element has been defined, the line of that previous element is assumed.

4.3.16 The LOWLIGHT Clause

Function

The LOWLIGHT clause specifies that the field is to appear on the screen with the lowest intensity. When only 2 levels of intensity are available, normal intensity and LOWLIGHT will be the same in the LOWLIGHT clause.

General Format

LOWLIGHT **E**

Syntax Rule

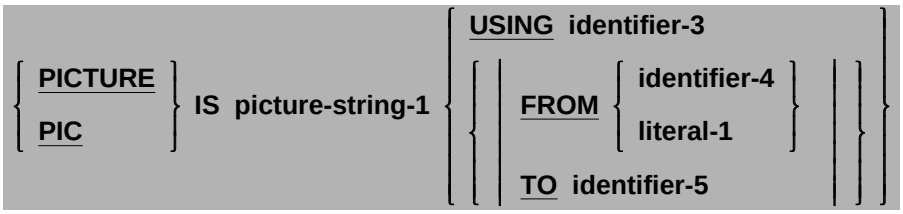
The LOWLIGHT clause may be specified only for an elementary screen description entry.

4.3.17 The PICTURE Clause

Function

The PICTURE clause describes the general characteristics and editing requirements of an elementary screen data element. It is also used to specify the data item whose contents are to be displayed when this element appears on the screen, and the data item in which the contents of this element are to be stored when it is accepted or modified.

General Format



Syntax Rule

Identifier-3, *identifier-4* and *identifier-5* shall be defined in the File, Working-Storage or Linkage Section.

General Rules

1. When data is to be transferred to the screen from one data item, possibly edited, and stored in a different data item, both the FROM and the TO phrases must be used in the PICTURE clause of the screen data element.
2. When data is to be transferred to the screen, possibly modified, and stored in the same data item (as when reading, modifying and rewriting records of a file), the USING phrase must be used in the PICTURE clause of the screen data element.
3. *Identifier-3*, *identifier-4*, *identifier-5* and *literal-1* need not be the same length as the screen data element to which the PICTURE clause applies. Data is transferred in accordance with the rules of the MOVE statement.
4. When the FROM phrase is specified:
   a. During the execution of the DISPLAY statement, data is transferred from *identifier-4* or *literal-1*, after being edited in accordance with *picture-string-1*, and displayed on the screen. The display begins at the screen position defined either implicitly or explicitly by the LINE and COLUMN clauses and the AT phrase of the DISPLAY statement.
   b. The FROM phrase has no meaning during the execution of an ACCEPT statement.

5. When the TO phrase is specified:

   a. Upon successfully completing the execution of the ACCEPT statement, data entered for the screen data element is transferred to *identifier-5*, after being edited in accordance with *picture-string-1*. The order in which data is transferred from the screen to screen data elements is the order the screen data elements are specified within the screen description of the screen item referenced by the ACCEPT statement.

   b. The TO phrase has no meaning during the execution of a DISPLAY statement.

6. When the USING phrase, or the FROM and the TO phrases are specified:

   a. During the execution of the DISPLAY statement, data is transferred from *identifier-3*, *identifier-4* or *literal-1* as described in rule 5a above.

   b. During the execution of the ACCEPT statement, data is transferred from *identifier-3*, *identifier-4* or *literal-1* as described in rule 5a above. Upon successfully completing the execution of the ACCEPT statement, data entered for the screen data element is transferred to *identifier-3* or *identifier-5* as described in rule 6a above.

For further rules refer to the PICTURE clause in ISO 1989:1985, The Nucleus Module.

Application Usage

Application writers are advised to specify numeric edited PICTURE character-strings for non-integer numeric screen items. It is advised not to use the symbol 'V', as the appearance of screen elements with an implied decimal point is implementation-dependent.

4.3.18 The REQUIRED Clause

Function

The REQUIRED clause specifies that in the context of an ACCEPT statement, the user must enter at least one character in the input or update field.

General Format

REQUIRED **E**

Syntax Rule

If the REQUIRED clause is specified at group level, it applies to each input and update field in that group.

General Rules

1. The REQUIRED clause takes effect during the execution of any ACCEPT statement that causes the screen item to be accepted, provided the cursor enters the screen item at some time during the accept operation. Unless this clause is satisfied, terminator keystrokes are rejected and the cursor is repositioned to the beginning of the item.
2. To satisfy this clause, alphanumeric screen items must contain at least one non-space character, and numeric screen items must have a non-zero value.
3. For update fields, the REQUIRED clause can be satisfied by the contents of the identifier or literal referenced in the FROM or USING phrase of the PICTURE clause, as well as by operator keyed data.
4. The REQUIRED clause may not be effective if a function key is used to terminate the accept operation.
5. The specification of the FULL and REQUIRED clauses together requires that the field must always be filled entirely by the user.
6. The REQUIRED clause is ignored for an elementary output field.

4.3.19 The REVERSE-VIDEO Clause

Function

The REVERSE-VIDEO clause specifies that the data is represented exchanging the default or specified foreground and background colours.

General Format

REVERSE-VIDEO **E**

Syntax Rule

The REVERSE-VIDEO clause may be specified only for elementary screen description entries.

4.3.20 The SECURE Clause

Function

The SECURE clause prevents data entered from the keyboard from appearing on the screen while the cursor is in the input field to which it applies.

General Format

SECURE **E**

Syntax Rule

The SECURE clause may only be specified for an input field.

General Rules

1. If the SECURE clause is specified at group level it applies to each input field in that group.
2. When the SECURE clause is used, characters entered for the input field do not appear on the screen.

4.3.21 The SIGN Clause

Function

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 ]                    E
            { TRAILING }
```

The rules for the SIGN clause are the same as in ISO 1989:1985, The Nucleus Module.

Application Usage

Application writers are advised to specify the SIGN clause with the SEPARATE CHARACTER option for all signed numeric screen items in order to avoid the implementation-dependent appearance of screen elements with the default sign encoding or any encoding without the SEPARATE CHARACTER option.

4.3.22 The UNDERLINE Clause

Function

The UNDERLINE clause specifies that each character of the field is underlined when it is displayed on the screen.

General Format

UNDERLINE **E**

Syntax Rule

The UNDERLINE clause may be specified only for elementary screen description entries.

4.3.23 The USAGE Clause

Function

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

General Format

[USAGE IS] DISPLAY **E**

The rules for the USAGE clause are the same as in ISO 1989:1985, The Nucleus Module.

4.3.24 The VALUE Clause

Function

The VALUE clause defines the constant value of a literal screen element.

General Format

VALUE IS literal-1 **E**

Syntax Rule

Literal-1 shall be a nonnumeric literal.

For further rules refer to the VALUE clause in ISO 1989:1985, The Nucleus Module.

4.4 PROCEDURE DIVISION

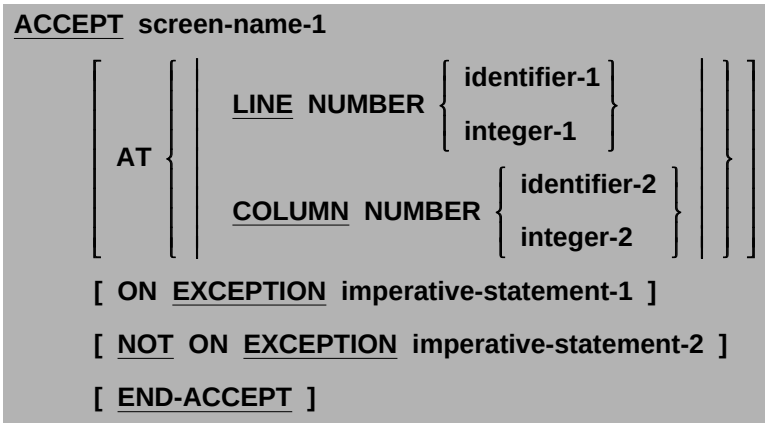
The Procedure Division statements that pertain to the screen handling facility are the ACCEPT and the DISPLAY statements.

4.4.1 The ACCEPT Statement

Function

The ACCEPT statement causes data to be transferred from the screen to the specified data items. This format of the ACCEPT statement is related to screen items defined in the Screen Section and allows enhanced screen handling facilities.

General Format



E

Syntax Rules

1. *Integer-1* and *integer-2* shall be unsigned.
2. *Identifier-1* and *identifier-2* shall be described as unsigned numeric data items.

General Rules

1. The AT phrase gives the absolute address on the screen where the execution of the ACCEPT statement is to start.
2. If the ON EXCEPTION phrase is specified, *imperative-statement-1* is executed if the execution of the ACCEPT statement finishes with anything other than a normal termination. If the NOT ON EXCEPTION phrase is specified, *imperative-statement-2* is executed if the execution of the ACCEPT statement finishes with a normal termination. Normal termination is defined in the definition of the CRT STATUS clause in the SPECIAL-NAMES paragraph.
3. The execution of the ACCEPT statement starts at line 1 if no line number is specified. It starts at column 1 if no column number is specified.

Application Usage

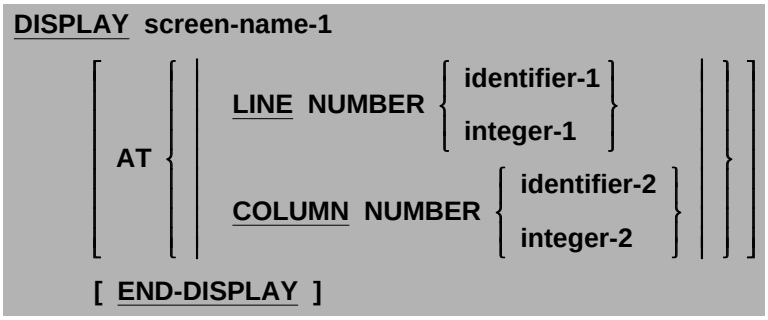
The AT phrase in combination with any LINE and COLUMN clauses specified in the screen described by *screen-name-1* should not be specified in such a way that screen items fall outside the LINES or COLS limits defined for the terminal concerned.

4.4.2 The DISPLAY Statement

Function

The DISPLAY statement causes data to be transferred from the specified data items to the screen. This format of the DISPLAY statement is related to screen items defined in the Screen Section and allows enhanced screen handling facilities.

General Format



E

Syntax Rules

1. *Integer-1* and *integer-2* shall be unsigned.
2. *Identifier-1* and *identifier-2* shall be described as unsigned numeric data items.

General Rules

1. The AT phrase gives the absolute address on the screen where the execution of the DISPLAY statement is to start.
2. The execution of the DISPLAY statement starts at line 1 if no line number is specified. It starts at column 1 if no column number is specified.

Application Usage

The AT phrase in combination with any LINE and COLUMN clauses specified in the screen described by *screen-name-1* should not be specified in such a way that screen items fall outside the LINES or COLS limits defined for the terminal concerned.

Chapter 5

File Sharing and Record Locking

5.1 INTRODUCTION

X/Open conforming systems have powerful multi-user and multi-process facilities, and these imply possible contention between processes for access to files and records within files. In order to control potential conflicts of file access between simultaneously running COBOL processes, certain defaults and COBOL syntax are provided. The defaults are entirely consistent with Standard COBOL, and are designed to allow COBOL applications developed for a single user to be used successfully in a multi-user X/Open environment without the need for any source code changes.

Each COBOL application accesses files through a file connector that is established via the ASSIGN clause. Although one file may be opened by one COBOL run unit more than once via different file connectors, this facility should not be used by X/Open conforming COBOL applications.

Where an X/Open conforming ISAM is used by the COBOL system to support indexed files, locking is defined to be consistent with other processes accessing files via the same X/Open conforming ISAM.

Locking is available for all file types: sequential, line sequential, relative and indexed.

5.2 CONCEPTS AT FILE LEVEL

Files are either active or inactive. An inactive file is one that is not open to any file connector. An active file is one that is open to one or more file connectors. Active files may be in one of two modes: exclusive or shareable.

A file which is in exclusive mode is open to one file connector only, and any other which attempts to open it receives a ''File locked'' error and is denied access. Exclusive mode implies that a file lock is held by the one file connector which is able to access the file; the file lock is released by that file connector closing the file. However, it is implementation-dependent whether exclusive mode will allow any number of file connectors specifying input mode to open that file.

A file that is in shareable mode is available to any number of file connectors that do not require exclusive use, each of which may protect data using the record locking concepts in the following paragraphs.

Active files should not be opened by another file connector in output mode. Files that are opened in output mode, and files that do not reside on mass storage devices, cannot be shared by X/Open conforming COBOL applications.

5.3 CONCEPTS AT RECORD LEVEL

Files opened in input mode are shareable by default. However, a file connector for a file opened in input mode will not acquire a lock.

Files opened in I-O or extend mode can be made shareable. While a file is opened in extend mode, other file connectors that open this file in input or I-O mode can read the complete logical file, but will not prevent the file connectors opened in extend mode from writing records by means of record locks.

5.3.1 Single, Automatic Record Locking

A file connector that has specified single record locking for a file (either explicitly or implicitly) can hold only one record lock in that file at any time. Single record locking specifies how record locks are released, and means that the system automatically releases an existing record lock whenever a new record is locked. This appears as one record lock that moves from record to record.

A record lock for a file is released by the same file connector taking any of the following actions:

- accessing any record in the file with any file operation except START
- executing an UNLOCK statement on that file
- closing the file

When the START statement is used, or any file operation that would acquire or release a record lock fails, the state of the lock for the accessed record is implementation-dependent. For this reason, X/Open conforming COBOL applications should issue an UNLOCK statement after such operations to ensure record locks are in a known state.

A file connector that specifies single record locking for a file should also specify automatic record locking for the file, which implies that it acquires a record lock whenever it reads a record in the file. Whenever the term automatic locking mode is used in this specification, it refers to automatic single record locking mode.

5.3.2 Multiple Record Locking

Locking of multiple records in a file is only available for relative or indexed files. A file connector that has specified multiple record locking for a file may hold a number of record locks for that file simultaneously. This prevents other file connectors accessing any member of the set of locked records, but will not deny them access to any records that are not locked.

All record locks for a file held for a file connector are released by the file connector taking either of the following actions:

- executing an UNLOCK statement on that file
- closing the file

A file connector that specifies multiple record locking for a file should also specify manual record locking for the file, which implies that it only acquires a record lock if it uses the READ statement with the WITH LOCK phrase. A READ without this phrase leaves any existing record locks for the file unchanged. Whenever the term manual

locking mode is used in this specification, it refers to manual multiple record locking mode.

The WRITE and REWRITE statements will not acquire record locks. This implies that files opened in extend mode will not lock any record. Acquiring locks by WRITE and REWRITE statements is addressed in **Appendix A**, **Future Directions**.

5.4 DEFAULT FILE LOCKS

The table below summarises the type of file lock that is held by a file connector for various types of files and their open modes.

File ORGANIZATION	Open Mode	Exclusive	Shareable		
			No Locks	Automatic	Manual
SEQUENTIAL	Input	S	D	-	-
	I-O	D	-	S	-
	Output	D	-	-	-
	Extend	D	S	-	-
LINE SEQUENTIAL	Input	S	D	-	-
	Output	D	-	-	-
	Extend	D	S	-	-
RELATIVE and INDEXED	Input	S	D	-	-
	I-O	D	-	S	S
	Output	D	-	-	-
	Extend	D	S	-	-

Where ‘‘D’’ represents the default, ‘‘S’’ represents selectable by syntax, and ‘‘-’’ represents an option not defined by X/Open.

5.5 I-O STATUS

Standard COBOL defines various values for the two character data item specified in the FILE STATUS clause. Two additional status conditions occur when locking files and records:

- OPEN fails; file locked, file not available for sharing, or not available for exclusive use
- READ with implicit or explicit lock, REWRITE or DELETE fails; record already locked via another file connector (implies a lock held by a separate process)

These two status conditions are represented by unique values for the FILE STATUS data item. These values have the leftmost character position set to a character value of 9 and the rightmost character position set to an implementation-dependent value. For truly portable programs, 88 level condition-names with the following specifications should be used in conjunction with the file status item (condition-names shown are examples):

```
88    FILE-LOCKED        VALUES "93", "9A".
88    RECORD-LOCKED      VALUES "99", "9D".
```

The actual values shown are representative for a number of implementations. However, it is no requirement on an implementation to support one of these values.

5.6 ENVIRONMENT DIVISION

5.6.1 The File Control Entry

Function

The file control entry names a file and allows specification of other file-related information. The LOCK MODE clause is an optional clause of the file control entry and is used to specify the locking technique used for the file.

General Format

Sequential File

```
SELECT [ OPTIONAL ] file-name-1
       ASSIGN TO { DISK          }                                  I
                 { PRINTER       }
                 { literal-1     }
                 { data-name-1   }                                  E
       [ [ ORGANIZATION IS ] SEQUENTIAL ]
       [ ACCESS MODE IS SEQUENTIAL ]
       [ LOCK MODE IS { AUTOMATIC [ WITH LOCK ON RECORD ] } ]     E
                      { EXCLUSIVE                         }
       [ FILE STATUS IS data-name-2 ] .
```

Line Sequential File

```
SELECT [ OPTIONAL ] file-name-1
       ASSIGN TO { DISK        }
                 { literal-1   }
                 { data-name-1 }
       [ ORGANIZATION IS ] LINE SEQUENTIAL                        E
       [ ACCESS MODE IS SEQUENTIAL ]
       [ LOCK MODE IS { AUTOMATIC [ WITH LOCK ON RECORD ] } ]
                      { EXCLUSIVE                         }
       [ FILE STATUS IS data-name-2 ] .
```

Relative File

```
SELECT [ OPTIONAL ] file-name-1
                                                                    I
      ASSIGN TO { DISK        }
                { literal-1   }
                { data-name-3 }                                     E
      [ ORGANIZATION IS ] RELATIVE
      [ ACCESS MODE IS { SEQUENTIAL [ RELATIVE KEY IS data-name-1 ]          } ]
                       { { RANDOM  } RELATIVE KEY IS data-name-1             }
                       { { DYNAMIC }                                         }
      [ LOCK MODE IS { MANUAL WITH LOCK ON MULTIPLE RECORDS      } ]      E
                     { AUTOMATIC [ WITH LOCK ON RECORD ]         }
                     { EXCLUSIVE                                 }
      [ FILE STATUS IS data-name-2 ] .
```

Indexed File

```
SELECT [ OPTIONAL ] file-name-1
                                                                    I
      ASSIGN TO { DISK        }
                { literal-1   }
                { data-name-4 }                                     E
      [ ORGANIZATION IS ] INDEXED
      [ ACCESS MODE IS { SEQUENTIAL } ]
                       { RANDOM     }
                       { DYNAMIC    }
      RECORD KEY IS data-name-1
      [ ALTERNATE RECORD KEY IS data-name-2 [ WITH DUPLICATES ] ] . . .
      [ LOCK MODE IS { MANUAL WITH LOCK ON MULTIPLE RECORDS      } ]      E
                     { AUTOMATIC [ WITH LOCK ON RECORD ]         }
                     { EXCLUSIVE                                 }
      [ FILE STATUS IS data-name-3 ] .
```

General Rules

1. The LOCK MODE clause specifies the locking technique to be used for the file unless it is overridden by the WITH LOCK phrase of the OPEN statement.
2. If the LOCK MODE clause is omitted, opening the file causes it to become exclusive, unless the file is opened for input. In the latter case, the file becomes shareable if the OPEN statement without the WITH LOCK phrase is successful.
3. When LOCK MODE IS EXCLUSIVE is specified, the file is opened in exclusive mode if the OPEN statement is successful.
4. When LOCK MODE IS AUTOMATIC or LOCK MODE IS MANUAL is specified, the file that is opened via the file connector is shareable if the OPEN statement without the WITH LOCK phrase is successful.
5. The WITH LOCK ON RECORD clause is for documentation purposes only.
6. When LOCK MODE IS AUTOMATIC [WITH LOCK ON RECORD] is specified for a file, a record lock is acquired by the successful execution of the READ statement and released on the successful execution of a subsequent I-O statement, with the exception of the START statement where the releasing of record locks is implementation-dependent.
7. When LOCK MODE IS MANUAL WITH LOCK ON MULTIPLE RECORDS is specified, a record lock is acquired by the READ WITH LOCK statement.
8. A file that is opened in exclusive mode cannot be opened by any other file connector. It is implementation-dependent whether a file that is opened in exclusive mode for input can be opened by another file connector if that OPEN statement specifies the INPUT mode too.
9. A file that is in shareable mode can be opened by any number of file connectors that do not require exclusive mode. It is implementation-dependent whether a file that is opened in shareable mode for input can be opened by another file connector in exclusive mode if that OPEN statement specifies the INPUT mode too.
10. The ASSIGN clause is described in **Section 8.3**, **File Assignment and Reassignment**.

Application Usage

A file that does not reside on a mass storage device should not be opened in shareable mode.

5.7 PROCEDURE DIVISION

5.7.1 The CLOSE Statement

General Rule

Following the successful execution of a CLOSE statement, all record or file locks held by the file connector on the closed file are released.

Application Usage

The WITH LOCK phrase has the meaning given in Standard COBOL, and hence shall not hold record or file locks acquired previously on the file.

5.7.2 The DELETE Statement

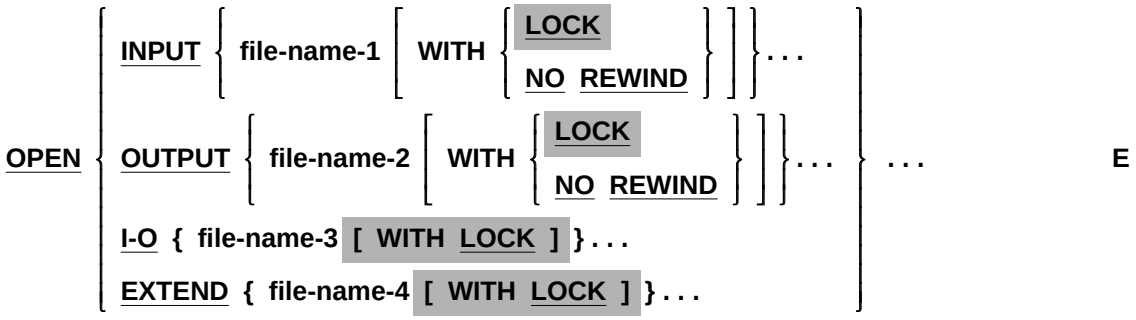
General Rules

1. A DELETE statement shall not be successful if any other file connector holds a lock on the record to be deleted.
2. Following the successful execution of the DELETE statement, any record lock held by the file connector on the deleted record is released.

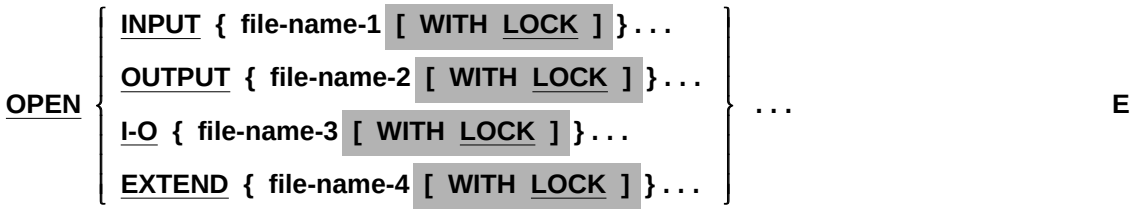
5.7.3 The OPEN Statement

General Formats

Sequential or Line Sequential File



Relative or Indexed File



General Rules

1. If the WITH LOCK phrase is specified, the OPEN statement acquires a lock on the whole file. (This is equivalent to specifying LOCK MODE IS EXCLUSIVE in the file control entry for the file.)
2. When LOCK MODE IS EXCLUSIVE is specified in the file control entry or the WITH LOCK phrase is specified in the OPEN statement, successful execution of the OPEN statement locks the file exclusively to that file connector. If the OPEN statement fails due to locking constraints, the file status shall be set to indicate ''file locked''.
3. When LOCK MODE IS AUTOMATIC or LOCK MODE IS MANUAL is specified and the WITH LOCK phrase is not specified in the OPEN statement, the file that is referred to is shareable. Such a file may be opened via more than one file connector. If the OPEN statement fails due to locking constraints, the file status shall be set to indicate ''file locked''.

5.7.4 The READ Statement

General Formats

Sequential, Line Sequential, Relative or Indexed File

```
READ file-name-1 [ NEXT ] RECORD [ INTO identifier-1 ]
     [ WITH [ NO ] LOCK ]                                    E
     [ AT END imperative-statement-1 ]
     [ NOT AT END imperative-statement-2 ]
     [ END-READ ]
```

Relative File

```
READ file-name-1 RECORD [ INTO identifier-1 ]
     [ WITH [ NO ] LOCK ]                                    E
     [ INVALID KEY imperative-statement-1 ]
     [ NOT INVALID KEY imperative-statement-2 ]
     [ END-READ ]
```

Indexed File

```
READ file-name-1 RECORD [ INTO identifier-1 ]
     [ WITH [ NO ] LOCK ]                                    E
     [ KEY IS data-name-1 ]
     [ INVALID KEY imperative-statement-1 ]
     [ NOT INVALID KEY imperative-statement-2 ]
     [ END-READ ]
```

General Rules

1. For files opened for input, READ or READ WITH LOCK statements will not acquire a record lock.
2. The READ statement, with either implicit or explicit lock, shall only be successful if no other file connector holds a lock on the record being accessed. If the record is locked by another file connector, the statement is unsuccessful, the file status is set to indicate ‘‘record locked’’ and, for a sequential READ, the setting of the file position indicator value is unaffected. For a random READ the setting of the file position indicator is unspecified following an attempt to read a record locked by another file connector.

3. When no next record exists at the time of execution of the READ statement, the AT END condition is returned regardless of any sharing of the file. This situation can, for example, occur if the file is opened in extend mode by another file connector.
4. If the file is opened for I-O and the lock mode is AUTOMATIC, each record is locked as it is read and released by the next I-O statement accessing the file connector, except as noted by the rules for the START statement. A READ statement with the WITH NO LOCK phrase shall not acquire a record lock.
5. If the file is opened for I-O and the lock mode is MANUAL, the file connector acquires a lock on the record only if the WITH LOCK phrase is specified. A READ statement without the WITH LOCK phrase shall not acquire a record lock.

Application Usage

Currently a READ statement without implicit or explicit lock, and this includes all READ statements for a file opened for input, may not detect a record lock (see **Appendix A**, **Future Directions**). Application writers should be aware of potential consistency problems in such situations.

Note that an attempt to reread a locked record may have the following effect:

a. The record is still locked or locked again.
b. If the record has been deleted:
   — the READ statement with KEY phrase returns an ‘‘invalid key’’ condition
   — the READ NEXT statement returns an ‘‘end of file’’ condition
   — the READ NEXT statement will attempt to read a different record than the previous READ statement
c. If the record is no longer the one that shall be selected based on the file position indicator because a new record has been inserted into the file, a READ NEXT will read the new record if that record is not locked.

The NO LOCK phrase should only be specified when records are being locked manually or automatically in a shareable file.

5.7.5 The REWRITE Statement

General Rules

1. A REWRITE statement shall not be successful if any other file connector holds a lock on the record to be rewritten.
2. If the lock mode is AUTOMATIC, a successful REWRITE statement releases an existing record lock.

5.7.6 The START Statement

General Rule

The START statement shall not acquire a record lock, nor will it detect a record lock. If the lock mode is AUTOMATIC, it is implementation-dependent whether it releases an existing record lock held by the same file connector.

Application Usage

A START statement followed directly by a READ NEXT statement may not read the record that was positioned at, and may possibly return an ‘‘end of file’’ or a ‘‘record locked’’ condition, as the record can be deleted or locked in the meantime. Similarly a READ statement followed directly by a START statement with KEY EQUAL or NOT LESS may not position on the record that was just read and may possibly return an ‘‘invalid key’’ condition.

5.7.7 The UNLOCK Statement

Function

The UNLOCK statement releases all record locks acquired by the file connector on a named file.

General Format

E

General Rules

1. The file referenced by file-name must already be successfully opened with the OPEN statement.
2. The UNLOCK statement is successful, even if no locks are currently held.

5.7.8 The WRITE Statement

General Rules

1. If two or more file connectors for a sequential file add records by sharing the file after opening it in extend mode the records are in an unspecified order.
2. 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.
3. If two or more file connectors for an indexed file add records by sharing the file after opening it in extend mode the order of alternate keys allowing for duplicates is unspecified.
4. If the lock mode is AUTOMATIC, a successful WRITE statement releases an existing record lock.

Application Usage

Even if the primary key values are unique and in ascending order, the WRITE statement may fail and return an ‘‘invalid key, sequence error’’ condition, when two or more file connectors share an indexed file after opening it in extend mode. Application writers should therefore open indexed files in extend mode with exclusive access.

Chapter 6

Internationalisation

6.1 INTRODUCTION

The internationalisation facilities defined by X/Open provide enhancements to the system that enable character-based data to be processed in a way that is independent of the underlying character encoding. This allows complete flexibility in the choice of coded character sets supported by an implementation. X/Open-compliant systems also allow program messages to be handled in the spoken language of each user, as well as supporting different formats for cultural data such as date/time formats and currency symbols. From the COBOL viewpoint, there are two mutually exclusive forms to support this:

- The complete displayable character set can be held within a single-byte encoding.

  Single-byte character sets are used for languages which are written using alphabet scripts such as Swedish, French or Russian; these character sets are relatively small. As single-byte national language encoding maps quite naturally onto the Standard COBOL alphanumeric data type, no syntax extensions are required. Only semantic extensions are required to support the X/Open internationalisation in COBOL.

- The complete displayable character set requires multi-byte encoding.

  Typically, multi-byte character sets are used for languages which are written with logographic (ideographic) scripts, such as Japanese, Korean or Chinese; these character sets are relatively large.

  The multi-byte character encoding represents a distinct new COBOL data type which requires extensions to Standard COBOL. This X/Open definition describes syntax and semantics for processing and handling multi-byte characters in a COBOL program. X/Open COBOL defines *national-characters* as multi-byte characters of a chosen coded character set that represents a very large displayable character set. This specification of multi-byte character support for COBOL can apply to various coded character sets.

  No semantic extensions are provided in this specification for internationalisation of multi-byte character sets (see **Appendix A**, **Future Directions**).

  This X/Open definition is modelled after the CODASYL COBOL Committee Proposal for National Character Handling Facility, the MIA Technical Requirements for the Programming Language COBOL, and common functionality in many existing COBOL implementations.

6.2 SINGLE-BYTE INTERNATIONALISATION SUPPORT

Single-byte internationalisation for COBOL is designed for use with languages which are written using alphabetic scripts in a way that is independent of the underlying character encoding. Where characters are to be transferred to or from terminal or printer devices, then, if required, any mapping of the character format will be done by the appropriate interface software.

The single-byte internationalisation support provides an automatic means of adapting a program to the character set, collating sequence and editing symbols associated with a particular country, without the need for the COBOL applications to know what these are in advance. The following features in a COBOL program are dependent on the particular country and character-set, and are selected at run time:

- class condition tests (ALPHABETIC, ALPHABETIC-UPPER and ALPHABETIC-LOWER)
- linguistic collating sequences in comparisons (for example, the correct collation of accented characters)
- linguistic collating sequences applied to indexed files
- linguistic collating sequences applied to sort or merge files
- character used for the currency symbol
- characters used for the decimal point and comma

6.2.1 Syntax Considerations

Single-byte internationalisation does not involve any special syntax. However, certain Standard COBOL facilities are potentially in conflict with the internationalisation facilities as defined here, and should not be used in X/Open conforming COBOL applications that are to use the internationalisation support. The following syntax should not be used:

- the PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph
- in the SPECIAL-NAMES paragraph:
  - — the ALPHABET clause
  - — the CURRENCY SIGN clause
  - — the DECIMAL-POINT IS COMMA clause
- the COLLATING SEQUENCE phrase in MERGE or SORT statements in the Procedure Division

The removal of the restrictions on the use of the above syntax is addressed in **Appendix A**, **Future Directions**.

The method of specifying at compile time that internationalisation support is required for a COBOL program, or at compile or run time that internationalisation support is required for a given indexed, sort or merge file, is currently implementation-dependent. X/Open addresses these problems in **Appendix A**, **Future Directions**.

6.2.2 Semantic Considerations

At run time the LANG environment variable must be set in order to obtain the semantics of COBOL internationalisation. This variable should be set before a COBOL application that uses internationalisation support is first executed, and should not be changed until the application completes. If the LANG environment variable is not set, then the semantics of Standard COBOL are used.

When a COBOL application executes with internationalisation support, the following operations shall be carried out in accordance with the settings of the locale specified in the LANG environment variable:

- nonnumeric relation conditions
- class condition tests
- key comparisons in indexed sequential files
- comparisons performed as part of a SORT or MERGE statement
- character used for the currency symbol
- characters used for the decimal point and comma

The operations related to files listed above are only effective when internationalisation support has been specified for the file.

For language settings that do not define the currency symbol as leading, Standard COBOL rules shall be observed for the position of the currency symbol. If the currency symbol is represented by more than one character, then only the first character is taken.

Nonnumeric relation conditions operate by the rules as defined in the selected collating sequence with the results being determined by the locale. Note, however, that comparisons and the MOVE operation comply with the Standard COBOL definition in which the logically shorter operands are padded with the figurative constant SPACE until both operands are of the same logical length.

Figurative constants (except the ALL literal) will not be affected by selecting internationalisation support, and will always refer to the same hexadecimal values as Standard COBOL without internationalisation support selected.

Figurative constants including the ALL literal will be expanded according to the rules of Standard COBOL prior to any internationalisation operation.

Class condition tests ALPHABETIC, ALPHABETIC-UPPER and ALPHABETIC-LOWER operate by the rules as defined in the selected class definition with the results being determined by the selected locale.

The class condition NUMERIC, and any user-defined class condition, will not be affected by selecting internationalisation support and will always refer to the same hexadecimal values as Standard COBOL, without internationalisation support selected.

Key comparisons in indexed files operate as described for string comparisons above and are language-dependent. X/Open conforming COBOL applications should access an indexed file in the same locale with the same collating sequence that was used when the file was created, unless the file is opened for output (see **Appendix A**, **Future Directions**). Locale and collating sequence are considered a fixed file attribute.

X/Open conforming COBOL applications that use internationalisation support should currently not use indexed, sort or merge files having variable length records (see **Appendix A**, **Future Directions**).

Comparisons performed as part of a SORT or MERGE statement shall be performed in a way consistent with key comparisons in indexed files.

The intrinsic functions CHAR, LOWER-CASE, MAX, MIN, NUMVAL, NUMVAL-C, ORD, ORD-MAX, ORD-MIN and UPPER-CASE operate by the rules as defined in the selected locale.

6.3 NATIONAL-CHARACTER SUPPORT

6.3.1 Terminology

The following terminology is used for national-character support:

Multi-byte character set
A character set composed of characters such that:

— every character is represented in a uniform size of *n* bytes, where *n* is greater than 1 and is implementation-dependent

— characters are represented in a non-uniform size of 2 to *n* bytes, where *n* is greater than 1 and is implementation-dependent

Internal-format
Representation of multi-byte characters in a uniform size and a manner suitable for internal processing by COBOL. Extraneous control information, if any, is not included in internal-format.

External-format
Representation of multi-byte characters such that they are displayable or printable. Control information, if any, necessary for displaying or printing is included in external-format.

6.3.2 National-character Support in a COBOL Source Program

The following character strings can be formed using multi-byte characters:

- COBOL words (user-defined words, system-names)
- literals
- comments

The following user-defined words can be formed using multi-byte characters:

- alphabet-name
- class-name
- condition-name
- data-name
- file-name
- index-name
- mnemonic-name
- paragraph-name
- record-name
- section-name
- symbolic-character

The following rules apply to user-defined words formed using multi-byte characters:

1. User-defined words containing multi-byte characters should contain only multi-byte characters.
2. The reference format, fixed or free form, determines the maximum length of user-defined words formed using multi-byte characters (see **Section 7.4**, **Limits**).
3. If the user-defined word containing multi-byte characters includes characters from the COBOL character set, then the following rules apply:
   a. The user-defined word may consist of multi-byte characters from the COBOL character set, and any other multi-byte characters. The only characters allowed from the COBOL character set are a-z, A-Z, 0-9 and the hyphen (-). The hyphen must not be used as the first or last character.
   b. The user-defined word may consist of only multi-byte characters from the COBOL character set. Users should not rely on any equivalence between single-byte and multi-byte characters of the COBOL character set. However, application writers need to be aware of possible future equivalence between the single-byte COBOL character set and the multi-byte character COBOL character set (see **Appendix A**, **Future Directions**).
   c. Although COBOL rules prescribe equivalence between upper-case and lower-case letters of the COBOL character set, users should not rely on this equivalence, or the distinction of the two, for user-defined words consisting of multi-byte characters (see **Appendix A**, **Future Directions**).
4. For maximum portability, application writers in a Japanese environment should only use KANJI, HIRAGANA and KATAKANA characters from the JIS character set in multi-byte character user-defined words.
5. User-defined words containing multi-byte characters must not be continued across lines.
6. Multi-byte characters are not allowed for data-names with the EXTERNAL clause, file-names with the EXTERNAL clause or the ASSIGN TO DISK clause.

System-names are implementation-dependent. The only system-name that can be specified in multi-byte characters is computer-name. The computer-name, however, is treated as documentation only.

No additional reserved words are introduced by national-character support for COBOL.

A national-character literal belongs to the category national-character. National-character literals have the following format:

```
N"national-character string"
```

Both of the characters that compose the beginning separator (N") for this literal must be represented in single-byte characters. The lower-case equivalent is also accepted (n"). Application writers should not embed quotation marks in national-character literals.

The reference format, fixed or free form, determines the maximum length of national-character literals (see **Section 6.3.3**, **Reference Formats** and **Section 7.4**, **Limits**).

The figurative constants SPACE(S), HIGH-VALUE(S) and LOW-VALUE(S) can be national-character literals when specified in the appropriate context. The figurative constant ALL national-character literal is also supported. The figurative constant SPACE/SPACES represents a sequence of one or more instances of the national-character space. The figurative constant HIGH-VALUE/HIGH-VALUES represents a sequence of one or more instances of the national-character that has the highest bit representation value. The figurative constant LOW-VALUE/LOW-VALUES represents a sequence of one or more instances of the national-character that has the lowest bit representation value.

Note that when culturally-sensitive collating sequences become standardised for multi-byte character sets, the HIGH-VALUE/LOW-VALUE figurative constants may be associated with different values. Therefore users should not rely on the value represented by the figurative constants HIGH-VALUE and LOW-VALUE.

National-character literals can be specified:

- in a VALUE clause associated with data items of class national-character or elementary screen item definitions
- in a MOVE statement as the sending item when a national-character or group item is the receiving item
- in a relation condition as an operand compared to a data item of class national-character or a group item
- wherever a nonnumeric literal is allowed, except:
  - — as a literal in the CALL statement when literal contains a program id
  - — as literals in the alphabet clause
  - — in the CURRENCY SIGN clause
  - — as the literal in the ASSIGN clause
  - — as the path-name-literal in the COPY statement
  - — as a literal in the CLASS clause
  - — as a literal in the CANCEL statement
  - — as an argument of any intrinsic function
  - — where restricted by further rules for specific statements

Multi-byte characters can be freely used in comments and comment-entries.

For national-character support in COBOL, one additional class and two additional categories are defined. The class national-character consists of the categories national-character and national-character-edited. To define an item as national-character:

- its PICTURE character-string must contain only the symbol ‘‘N’’
- its content, when represented in standard data format, must be one or more national-characters

To define an item as national-character-edited:

- its PICTURE character-string must contain only the symbols ‘‘N’’ and ‘‘B’’, where each ‘‘B’’ in the character-string represents a character position into which the national-character space is inserted and is counted in the size of the item
- its content, when represented in standard data format, must be two or more national-characters

Each ‘‘N’’ in a PICTURE character-string represents the amount of physical storage required to store a single national-character. Each ‘‘B’’ in a PICTURE character-string represents the amount of physical storage required to store a single national-character space. Both PICTURE symbols represent a fixed amount of storage. Further characteristics of the physical storage are implementation-dependent. In a PICTURE character-string, lower-case ‘‘n’’ and ‘‘b’’ symbols are equivalent to their corresponding upper-case symbols ‘‘N’’ and ‘‘B’’.

The BLANK WHEN ZERO clause must not be specified for data items of class national-character.

The JUSTIFIED RIGHT clause can be specified for national-character data items. When JUSTIFIED RIGHT is specified for a national-character receiving data item, and the sending data item is larger than the receiving data item, the leftmost characters are truncated. When JUSTIFIED RIGHT is specified for a national-character receiving data item that is larger than the sending data item, the data is aligned in the receiving data item at the rightmost character position, and any unused positions on the left are filled with spaces.

The OCCURS clause can be specified for data items of class national-character. A data item of class national-character can be specified in the ASCENDING / DESCENDING KEY phrase in the OCCURS clause for use in the SEARCH ALL statement only. ASCENDING KEY and DESCENDING KEY national-character items, specified in the OCCURS clause and the SEARCH ALL statement, are used for comparisons based on the binary collating sequence of the values of the national-characters.

The REDEFINES clause and the RENAMES clause should not be used for data items of class national-character. The effects of using either clause for national-character items may not be portable.

The SYNCHRONIZED clause is ignored for national-character data items.

A VALUE clause associated with a data item of class national-character must contain either a national-character literal or the figurative constant SPACE, HIGH-VALUE or LOW-VALUE.

In a condition-name entry, if the associated data item is of class national-character, all literals in the VALUE clause must be national-character literals or the figurative constants SPACE, HIGH-VALUE or LOW-VALUE. The range of national-character literals specified for the THROUGH phrase is based on the binary collating sequence of the values of the national-characters.

6.3.3 Reference Formats

X/Open COBOL defines two alternate reference formats that can be used for COBOL source programs: fixed form and free form.

- *Fixed form reference format* refers to the reference format defined in Standard COBOL. When using fixed form reference format, X/Open COBOL defines the following rules for national-character data in the source program.

  The maximum number of bytes allowed in a source line preceding margin R is 72 bytes. The Sequence Number Area occupies 6 bytes. National-characters are not allowed in this area, nor in the Indicator Area (1 byte) that immediately follows the Sequence Number Area. A national-character user-defined word should not contain more than 29 characters. A national-character literal should not contain more than 18 characters. A national-character literal should not be continued on one or more continuation lines.

- *Free form reference format* is modelled after the CODASYL COBOL reference format for free form source. The maximum number of characters allowed in a source line is 80 characters. In contrast to fixed form, free form reference format does not impose any additional restrictions on the maximum length of multi-byte character user-defined words and national-character literals, i.e., multi-byte character user-defined words can be up to 30 characters in length and national-character literals can be up to 160 characters in length (see **Section 3.8**, **Free Form Reference Format**, and **Section 7.4**, **Limits**).

6.3.4 National-character Support in COBOL Operations

Data items of class national-character can be specified in the following COBOL operations:

- RECORD KEY clause, ALTERNATE RECORD KEY clause

  Record key can specify a data item of class national-character. When a record key of class national-character is specified, a KEY specified in a READ or START statement must also be of class national-character.

  Note that RECORD KEY and ALTERNATE RECORD KEY items of class national-character are used for comparisons based on the binary collating sequence of the values of the national-characters (see the following section **Relation Conditions and Reference Modification**).

- ACCEPT statement

  A data item of class national-character can be specified as the data item into which data is accepted. Mnemonic-name must be associated with SYSIN, ENVIRONMENT-VALUE or ARGUMENT-VALUE.

- CALL...USING statement

  Data items in the USING phrase can be of class national-character. If operands in the USING phrase of the Procedure Division header are of class national-character, then the corresponding operands in the USING phrase of the CALL statement must also be of class national-character.

- COPY...REPLACING statement

  Either operand of the REPLACING phrase can be:

  — pseudo-text containing national-characters

  — a data item or literal of class national-character

  — a user-defined word formed using national-characters

  Both characters that compose a pseudo-text delimiter (==) must be represented in single-byte characters.

- DISPLAY statement

  Data items and literals of class national-character can be displayed. Mnemonic-name must be associated with SYSOUT, SYSERR or ENVIRONMENT-VALUE.

- EVALUATE statement

  Where identifiers are permitted, they may reference data items of class national-character. Where nonnumeric literals are permitted, national-character literals are permitted.

- INITIALIZE statement

  Identifier-1 can specify a data item of class national-character, or a group item that contains subordinate items of class national-character. The REPLACING phrase should not be specified. The figurative constant SPACE is the implied sending field for data items of class national-character.

- INSPECT statement

  Identifier-2 must not be of class national-character. If any identifiers or literals other than *identifier-2* are of class national-character, then all of them must be of class national-character. The count maintained in *identifier-2* tallies national-characters, not bytes.

- MERGE statement

  The KEY phrase may specify a data item of class national-character.

- MOVE statement

  If the sending data item is of class national-character, then all receiving data items must be of class national-character, or all receiving data items must be group data items. If any receiving data item is of class national-character, then all receiving data items must be of class national-character, and the sending item may either be of class national-character or be a group item. When a group item is a sending or receiving item, the alphanumeric MOVE rules apply, and when necessary, data is padded on the right with single-byte character spaces when stored in the receiving item. National-character literals, and the figurative constants SPACE, HIGH-VALUE and LOW-VALUE, can also be specified as sending items. If a receiving and sending item are of class national-character and of unequal size, the data will either be truncated or padded in accordance with the rules of COBOL with national-character spaces when stored in the receiving item. If a receiving item is a national-character-edited data item, editing will occur as specified.

- READ...INTO statement

  Identifier-1 can be a data item of class national-character. If both *identifier-1* and the only record description entry subordinate to *file-name-1* are of class national-character, national-character MOVE rules shall apply, otherwise alphanumeric MOVE rules shall apply (see the MOVE statement above). For indexed files, the KEY phrase may specify a data item of class national-character.

- REPLACE statement

  Pseudo-text can contain multi-byte characters and national-characters. Both of the characters that compose a pseudo-text delimiter (==) must be represented in single-byte characters.

- REWRITE...FROM statement

  Identifier-1 can be a data item of class national-character.

- SEARCH (ALL) statement

  Identifier-1 can be a data item of class national-character. In a SEARCH ALL statement, if *identifier-1* is a data item of class national-character, then the ASCENDING / DESCENDING KEY item defined in its OCCURS clause must also be of class national-character.

- SET...TO TRUE statement

  If *condition-name-1* is associated with a data item of class national-character, the SET...TO TRUE statement is permitted.

- SORT statement

  The KEY phrase may specify a data item of class national-character.

- START...KEY statement

  For indexed files, the KEY phrase may specify a data item of class national-character.

- STRING statement

  Identifier-4 must not be of class national-character. If any identifiers or literals other than *identifier-4* are of class national-character, then all of them must be of class national-character. The *identifier-4* indicates the relative character position, rather than the relative byte position, in the receiving field.

- UNSTRING statement

  If *identifier-1*, *identifier-2*, *identifier-3*, *literal-1* or *literal-2* are of class national-character, then all of them must be of class national-character. *Identifier-4* and *identifier-5* can be data items of class national-character. The count maintained in *identifier-6* tallies national-characters, not bytes. *Identifier-7* indicates the relative character position, rather than the relative byte position, in the receiving field.

- WRITE...FROM statement

  Identifier-1 can be a data item of class national-character.

Relation Conditions and Reference Modification

Data items and literals of class national-character can be compared with:

- other data items and literals of class national-character
- the figurative constants SPACE, HIGH-VALUE, LOW-VALUE
- group items

Relation conditions are based on the binary collating sequence of the values of the national-characters and not on locale settings. If the national-character operands are not the same length, the smaller operand is padded on the right with national-character spaces. The alphanumeric comparison rules are applied to comparisons between a national-character operand and a group item. (If a national-character operand is compared with a group item, the national-character operand will be treated as though it were moved to a group item of a length equivalent to the number of bytes occupied by the national-character item, and this were then compared with the original group item.)

Note that the PROGRAM COLLATING SEQUENCE clause does not apply to comparisons between national-character operands.

Reference modification shall be permitted for data items of class national-character. Reference modification refers to characters and length in characters, not bytes. The unique data item created by reference modification of a national-character or national-character-edited data item is considered to be of the class and category national-character.

I-O

When ORGANIZATION IS LINE SEQUENTIAL or ASSIGN TO PRINTER is specified in the file control entry, the following rules apply for national-character-data:

1. READ statement (for line sequential files only): if the record area references a data item of class national-character, conversion of data from external-format to internal-format will occur before the data is stored in the record. Only one record description entry should be subordinate to the associated file description entry (FD). The effect of using multiple record description entries may not be portable.
2. WRITE statement: if *record-name-1* is of class national-character, or contains elementary data items of class national-character, conversion of data from internal-format to external-format will occur before a logical record is released.

In the same way, the following rules apply for national-character data in the standard ACCEPT and DISPLAY statements.

1. ACCEPT statement: data that is input to the ACCEPT statement will be converted from displayable-format to internal-format before it is stored in the national-character data item.
2. DISPLAY statement: data that is output from the DISPLAY statement will be converted from internal-format to displayable-format before it is transferred to the hardware device.

Screen Handling

Each elementary item in a screen definition that contains a VALUE clause specifying a national-character literal defines a national-character screen data element. Each elementary item in a screen definition that contains a PICTURE character-string containing the symbol ‘‘N’’, and optionally also ‘‘B’’, defines a national-character screen data element.

The following rules apply for national-character screen data elements in the ACCEPT and DISPLAY statements:

1. ACCEPT statement: data that is input to the ACCEPT statement will be converted from displayable-format to internal-format before it is stored in the national-character data item.
2. DISPLAY statement: data that is output from the DISPLAY statement will be converted from internal-format to displayable-format before it is transferred to the hardware device.

See **Chapter 4**, **Screen Handling Module** for background.

Command Line Arguments and Environment Variables

The following rules apply for national-character data retrieved from the command line or an environment variable, or stored in an environment variable:

1. ACCEPT statement: if *identifier-2* of the ACCEPT statement is a data item of class national-character, data will be converted from displayable-format to internal-format before it is stored in the data item.
2. DISPLAY statement: if *identifier-2* or *literal-2* of the DISPLAY statement is a data item of class national-character, data will be converted from internal-format to external-format before it is transferred to the environment variable.

See **Section 8.6**, **Access to Command Line Arguments and Environment Variables** for background.

Chapter 7

Portability Issues

7.1 RESTRICTIONS FOR PORTABLE PROGRAMS

7.1.1 Arithmetic Expressions

The techniques used in handling arithmetic expressions are implementation-dependent. Therefore, application writers should be aware that the results may differ across implementations, and should avoid using arithmetic expressions when maximum portable results are desired.

Application writers should restrict themselves to using integer exponents.

7.1.2 Memory Allocation

The manner in which memory is allocated for called programs may differ between interpreted and fully compiled implementations. The management of memory required for a series of called programs should not be defined by the application developer, but memory should be allocated dynamically at run time. This method of memory allocation will offer maximum portability but could cause memory fragmentation. Additionally, developers should not make any assumptions regarding the contents of variables on second and subsequent calls to a program for cases that are not specifically defined in Standard COBOL.

7.1.3 Intrinsic Functions

The ACOS, ANNUITY, ASIN, ATAN, COS, LOG, LOG10, MEAN, PRESENT-VALUE, SIN, SQRT, STANDARD-DEVIATION, TAN and VARIANCE functions will give approximate results by their nature. Application writers should not assume that the results of any of these functions for the same argument(s) are identical across implementations. Moreover, the precision of the result should not be assumed to exceed 16 decimal digits.

In addition, many of the intrinsic functions may have intermediate results that are implementation-dependent. In particular the MEAN, SUM, STANDARD-DEVIATION and VARIANCE functions may cause arithmetic overflow (see **Arithmetic Expressions** above).

7.2 EXCLUSIONS

This section summarises the exclusions from Standard COBOL in a convenient form for those familiar with it. It describes the exclusions in relation to the ''modules'' defined in Standard COBOL.

- Complete modules that are in Standard COBOL, but are excluded from the X/Open definition:
  - — communication
  - — debug
  - — report writer
  - — segmentation
- Obsolete elements of Standard COBOL included in a conforming X/Open implementation which conforms to the high subset of Standard COBOL that should not be used in a conforming X/Open COBOL source program:
  - — double character substitution (Nucleus module)
  - — ALL literal greater than one digit associated with a numeric or numeric edited item (Nucleus module)
  - — all commentary items in the Identification Division (Nucleus module)
  - — MEMORY SIZE clause (Nucleus module)
  - — RERUN clause (Sequential, Relative and Indexed I-O modules)
  - — MULTIPLE FILE TAPE clause (Sequential I-O module)
  - — LABEL RECORDS clause (Sequential, Relative and Indexed I-O modules)
  - — VALUE OF clause (Sequential, Relative and Indexed I-O modules)
  - — DATA RECORDS clause (Sequential, Relative, Indexed I-O and Sort-Merge modules)
  - — ALTER statement (Nucleus module)
  - — ENTER statement (Nucleus module)
  - — GO TO statement without procedure-name (Nucleus module)
  - — REVERSED phrase of the OPEN statement (Sequential I-O)
  - — STOP LITERAL statement (Nucleus module)
- Further elements of Standard COBOL included in a conforming X/Open implementation that should not be used in a conforming X/Open COBOL source program as their use in programs will prevent portability:
  - — ellipses in ASSIGN clause (Sequential, Relative, Indexed I-O and Sort-Merge modules)
  - — mnemonic-name in ADVANCING phrase of WRITE statement (Sequential I-O module)

— BLOCK CONTAINS clause (Sequential, Relative and Indexed I-O modules)

— CODE-SET clause (Sequential I-O module)

— PADDING CHARACTER clause (Sequential, Relative and Indexed I-O modules)

— CONTAINS *integer-1* to *integer-2* option of the RECORD clause (Sequential, Relative, Indexed I-O and Sort-Merge modules)

— RECORD DELIMITER clause (Sequential I-O module)

— RESERVE clause (Sequential, Relative and Indexed I-O modules)

— SYNCHRONIZED clause (Nucleus module)

7.3 OBSOLETE ELEMENTS

The X/Open COBOL elements discussed in this section are obsolete elements. This means that X/Open intends to remove these elements from a future edition of this specification. X/Open conforming implementations are required to support these elements. However, application writers should not use them but should instead use the alternatives specified in this section.

7.3.1 Continuation Using the Hyphen

The continuation of lines using the hyphen (-) in the indicator area of the continuation line, possibly in combination with a nonnumeric literal without a closing quotation mark on the continued line, as defined in Standard COBOL, is an obsolete element in this specification.

There is no real need for continuation of COBOL words, numeric literals or PICTURE character strings as each one easily fits on one line.

The concatenation of nonnumeric literals (specified in **Section 3.7**, **Concatenation Expressions**) should be used instead of the continuation of such literals.

7.4 LIMITS

This section defines the limits that all X/Open-compliant COBOL implementations shall meet or exceed. For example, the maximum length of a group item may vary between 65000 and several MBytes depending on the COBOL implementation. For maximum portability, developers should restrict themselves to the limits listed in this section.

As any COBOL Compiler System has some hidden restrictions which cannot be expressed in terms of COBOL syntactic constructs, but are implicitly defined by many parameters, this list is not complete. That means, even programs that are below all of these limits may run out of limits defined only by a combination of many restrictions. Limits that are defined in Standard COBOL are not mentioned.

The following list is divided into two parts. The first gives restrictions in terms of COBOL syntax, that at least one of the regarded products has explicitly specified. The second list gives some restrictions that might be influenced by the operating system or that do not have impact on the COBOL source itself, but only on the run-time environment.

Compile-time Limits

Fixed Form Reference Format	
Number of bytes preceding margin R	72
Size of national-character user-defined word	29 characters
Size of national-character literal	18 characters
Free Form Reference Format	
Number of characters in a source line	80
Number of bytes in a source line	255
Nucleus	
Size of alphanumeric items	32767 bytes
Size of numeric edited item	30 bytes
Size of group items	65000 bytes
Size of tables	65000 bytes
Number of indexes per table	12
Number of KEY IS phrases	12
Size of key per table	100 bytes
Number of subscripts	7
Number of labels in GO TO DEPENDING	99
Number of nesting levels in IF statement	25
Dynamic nesting of simple PERFORM	100
Dynamic nesting of PERFORM n TIMES	22
Number of WHEN phrases in SEARCH	12
Number of delimiters in UNSTRING	30
Number of identifiers in INSPECT	15
Number of levels of nesting parentheses in arithmetic expression	15
Characters in ISO ACCEPT DISPLAY	220
Maximum value of integers in formats	65000

Replication count in PICTURE string	32767
Total length of literals	32767 bytes
Inter-Program Communication	
Number of parameters in Procedure Division USING	62
Number of parameters in CALL...USING	62
Length of a program-name	8 characters
Length of external data-names and file-names (See **Section 8.10**, **External Names and Case Conventions** for further restrictions.)	8 characters
All I-O modules	
Size of one record	2000 bytes
Number of SAME clauses	10
Length of a file-name when assigned to DISK, including the extensions added by the X/Open Environment	14 characters
Indexed I-O	
Size of key	100 bytes
Number of alternate keys	15
Sort-Merge	
Number of Sort or Merge input files	12
Number of keys in SORT or MERGE	12
Size of key	100 bytes
Intrinsic Functions	
Number of explicit arguments	62

Run-time Limits

Nucleus	
Size of environment variables	127 characters
Size of command line	128 characters
All I-O	
External file-name including pathname	100 characters
Number of files open at one time (indexed files counting as two)	50
Number of file and record locks at one time	†
Relative I-O	
Maximum value of a relative key	2 147 483 647

† Dependent on operating system setting and number of simultaneous processes and files.

Chapter 8

COBOL in an X/Open Environment

8.1 INTRODUCTION

This chapter addresses the special characteristics of X/Open-compliant systems, and the effects these have on the functionality of COBOL compilers. It includes recommended techniques for ensuring the maximum portability of COBOL programs.

8.2 FILES

The COBOL system shall set the permissions, ownership, etc., for files in a manner consistent with X/Open conventions.

The COBOL run-time system is configurable to support an ISAM file handling package that conforms to the X/Open ISAM definition (see **Referenced Documents**).

8.3 FILE ASSIGNMENT AND REASSIGNMENT

In Standard COBOL, the interpretation of the argument of the ASSIGN clause is implementation-dependent.

General Format

Syntax Rule

Data-name-1 may not be equal to one of the implementor-names DISK or PRINTER.

General Rules

1. ASSIGN TO DISK implies a disk file using *file-name-1* as the file-name in the current directory.
2. ASSIGN TO PRINTER implies a print file. A print file is a file that can be printed. It should only contain printable characters and line and page advancing information. The ADVANCING phrase of the WRITE statement should only be used in conjunction with print files.
3. ASSIGN TO *literal-1* implies a disk file whose access path is specified by the literal.
4. ASSIGN TO *data-name-1* implies a disk file whose access path is given via the contents of the data item referenced by *data-name-1*.

The ability to reassign a file-name at run time, via an environment variable by means of an ACCEPT statement, is described in **Section 8.6**, **Access to Command Line Arguments and Environment Variables**.

Application Usage

Application writers are recommended to adopt the use of names relative to the current directory to facilitate implementation in a variety of systems.

8.4 SPECIAL FILES

The X/Open COBOL definition includes provision for special files SYSIN, SYSOUT and SYSERR. If the FROM phrase in the ACCEPT statement is omitted, the default is SYSIN, and if the UPON phrase in the DISPLAY statement is omitted, the default is SYSOUT. SYSIN, SYSOUT and SYSERR are associated with the standard input, standard output and standard error devices (*stdin*, *stdout* and *stderr*) respectively.

Simple terminal handling can be achieved by using the standard ACCEPT and DISPLAY statements which access standard input and output respectively. More sophisticated terminal handling is appropriate with cursor addressable terminals, and COBOL makes this available in a terminal-independent way using the ACCEPT and DISPLAY statements in conjunction with the Screen Section. By including this syntax within the COBOL program, standard input and output are passed through the COBOL screen handling filter, which uses, or is compatible with, the X/Open curses interface (see the X/Open **Terminal Interfaces** specification listed in **Referenced Documents**). In portable applications using ACCEPT or DISPLAY *screen-name*, the standard input and standard output devices (*stdin* and *stdout*) should not be redirected.

8.5 ERROR REPORTING

The default output device for both compile-time and run-time errors shall be the (possibly redirected) standard error device.

The application programmer may write to the standard error device from within a COBOL program by using the DISPLAY statement, as described in **Section 8.4**, **Special Files**.

8.6 ACCESS TO COMMAND LINE ARGUMENTS AND ENVIRONMENT VARIABLES

Function

The following formats of the ACCEPT statement retrieve the number of arguments on the command line and the value of a command line argument or an environment variable.

The following formats of the DISPLAY statement specify the position of an argument within the command line, the name of an environment variable and store a value in the specified environment variable.

General Format

```
SPECIAL-NAMES .
    ARGUMENT-NUMBER   IS mnemonic-name-1
    ARGUMENT-VALUE    IS mnemonic-name-2
    ENVIRONMENT-NAME  IS mnemonic-name-3
    ENVIRONMENT-VALUE IS mnemonic-name-4 .
```

I

Format 1 (READ the Number of Arguments that Appear on the Command Line)

```
ACCEPT identifier-1 FROM mnemonic-name-1 [ END-ACCEPT ]
```

E

Format 2 (READ the Arguments from the Command Line One After Another)

```
ACCEPT identifier-2 FROM mnemonic-name-2
       [ ON EXCEPTION imperative-statement-1 ]
       [ NOT ON EXCEPTION imperative-statement-2 ]
       [ END-ACCEPT ]
```

E

Format 3 (READ an Argument from the Command Line by its Position)

```
DISPLAY { identifier-3 } UPON mnemonic-name-1 [ END-DISPLAY ]
        { integer-1    }
```

E

```
ACCEPT identifier-2 FROM mnemonic-name-2
       [ ON EXCEPTION imperative-statement-1 ]
       [ NOT ON EXCEPTION imperative-statement-2 ]
       [ END-ACCEPT ]
```

E

Format 4 (READ from an Environment Variable)

```
DISPLAY { identifier-4 } UPON mnemonic-name-3 [ END-DISPLAY ]                E
        { literal-1    }
ACCEPT identifier-2 [ FROM mnemonic-name-4 ]
      [ ON EXCEPTION imperative-statement-3 ]                                E
      [ NOT ON EXCEPTION imperative-statement-2 ]
      [ END-ACCEPT ]
```

Format 5 (WRITE to an Environment Variable)

```
DISPLAY { identifier-4 } UPON mnemonic-name-3 [ END-DISPLAY ]                E
        { literal-1    }
DISPLAY { identifier-2 } UPON mnemonic-name-4
        { literal-2    }
      [ ON EXCEPTION imperative-statement-4 ]                                E
      [ NOT ON EXCEPTION imperative-statement-2 ]
      [ END-DISPLAY ]
```

Syntax Rules

1. *Identifier-1* and *identifier-3* must reference a data item described as an unsigned integer.
2. *Identifier-2* and *identifier-4* must reference an alphanumeric data item.
3. *Integer-1* must be unsigned.
4. *Literal-1* and *literal-2* must be nonnumeric literals.

General Rules

1. The implementor-name ARGUMENT-NUMBER, if used with a format 1 ACCEPT, retrieves into *identifier-1* the number of arguments contained in the command line. When used with a format 3 DISPLAY, it sets the current argument position indicator for the command line to *integer-1* or the contents of *identifier-3*. The current argument position indicator determines which command line argument value a format 2 or format 3 ACCEPT statement returns into *identifier-2*, as specified in the following general rules.
2. Without a format 3 DISPLAY, the initial value of the current argument position indicator is 1.
3. The current argument position indicator is incremented by 1 after execution of a format 2 or format 3 ACCEPT statement.

4. The implementor-name ARGUMENT-VALUE, when used in a format 2 or format 3 ACCEPT, retrieves the value of the current argument as indicated by the current argument position indicator.

5. The implementor-name ENVIRONMENT-NAME, if used in a DISPLAY shown in format 4 or format 5, sets the name of an environment variable to *literal-1* or the contents of *identifier-4*.

6. The implementor-name ENVIRONMENT-VALUE, if used in a format 4 ACCEPT, retrieves the value of the environment variable (named as specified in general rule 5) into *identifier-2*. If used in a format 5 DISPLAY, it sets the contents of the environment variable (named as specified in general rule 5) to *literal-2* or contents of *identifier-2*.

7. The COBOL system shall supply any control information needed by the operating system to select the proper environment variable and to set its contents. This may include a terminating character.

8. *Integer-1* or the contents of *identifier-3* must be in the range 0 to 99, and may refer to arguments, switches and flags which appear on the command line of the COBOL program. When the current argument position indicator is zero, it refers to the zeroth command line argument, i.e., the command that invoked the COBOL program.

9. *Imperative-statement-1* is executed if in format 2 an attempt is made to read beyond the last argument on the command line, or if in format 3 the argument does not exist.

10. *Imperative-statement-3* is executed if the name of the environment variable has not been set via a format 4 DISPLAY, or if the environment variable does not exist.

11. *Imperative-statement-4* is executed if the name of the environment variable has not been set via a format 5 DISPLAY, or if not enough space could be allocated to store the environment variable.

12. *Imperative-statement-2* is executed if the exception condition does not exist.

13. The effect of retrieving command line arguments and of retrieving the number of arguments in a program that is called by another program, is implementation-dependent.

Example

```
SPECIAL-NAMES.
    SYSERR                IS STANDARD-ERROR
    ENVIRONMENT-NAME      IS NAME-OF-ENVIRONMENT-VARIABLE
    ENVIRONMENT-VALUE     IS ENVIRONMENT-VARIABLE
    ARGUMENT-NUMBER       IS POS-OF-COMMAND-LINE-ARGUMENT
    ARGUMENT-VALUE        IS COMMAND-LINE-ARGUMENT.

PROCEDURE DIVISION.

    ACCEPT file-length FROM COMMAND-LINE-ARGUMENT
        ON EXCEPTION
            DISPLAY "No arguments specified"
                UPON STANDARD-ERROR
```

```
        END-DISPLAY
        MOVE –1 TO RETURN-CODE
        STOP RUN
END-ACCEPT.

DISPLAY "COBOLPATH" UPON NAME-OF-ENVIRONMENT-VARIABLE.
ACCEPT env-dir FROM ENVIRONMENT-VARIABLE
    ON EXCEPTION
        DISPLAY "Environment COBOLPATH is not set"
            UPON STANDARD-ERROR
        END-DISPLAY
    NOT ON EXCEPTION
        ACCEPT file-name FROM COMMAND-LINE-ARGUMENT
            ON EXCEPTION
                DISPLAY
                "Attempt to read beyond end of command line"
                    UPON STANDARD-ERROR
                END-DISPLAY
            NOT ON EXCEPTION
                STRING env-dir, "/", file-name ... INTO ...
                OPEN ...
        END-ACCEPT
END-ACCEPT.

DISPLAY "TZ" UPON NAME-OF-ENVIRONMENT-VARIABLE.
ACCEPT old-TZ FROM ENVIRONMENT-VARIABLE.
DISPLAY "UTC0" UPON ENVIRONMENT-VARIABLE
    ON EXCEPTION
        DISPLAY "Environment space exhausted"
            UPON STANDARD-ERROR
        END-DISPLAY
        MOVE –1 TO RETURN-CODE
        STOP RUN
END-DISPLAY.
```

Assume the environment is set to the following:

```
$ COBOLPATH=/usr/files
$ TZ=MET–1MET
```

Then, during execution of:

```
$ cblprog 2000 mydata
```

the data items:

- file-length will contain "2000"
- env-dir will contain "/usr/files"
- file-name will contain "mydata"
- old-TZ will contain "MET-1MET"

and the environment variable TZ will be set to TZ=UTC0.

Application Usage

The environment name and environment value have the same size as the COBOL literal or identifier specified in the appropriate DISPLAY statements. Application writers should use reference modification to match the length of the identifier to the length of the value to be set.

8.7 CALLING OTHER LANGUAGES

Routines written as C language functions (and other languages that conform with the C argument passing conventions) can be compiled to object modules and be linked to the COBOL module to allow direct calling by name of both C by COBOL programs and COBOL by C functions in any combination. The main program can also be either COBOL or C.

A CANCEL statement referencing a program not written in COBOL has an implementation-dependent effect.

Data items in a COBOL program for which the EXTERNAL clause is specified, and identifiers in a C compilation unit having external linkage, shall refer to the same object if their names are identical (see **Section 8.10**, **External Names and Case Conventions**).

8.8 COBOL DATA TYPES

In order to support data types for X/Open-compliant SQL, ISAM and C Language interfaces, BINARY (machine-independent) and COMP-5 (machine-dependent) have been defined. BINARY and COMP-5 are defined in **Chapter 3**, **Definition of Extensions**.

8.9 RETURN CODE

There is a special register automatically defined within each COBOL run-unit with the name RETURN-CODE. This special register may be set by a program prior to the execution of a STOP RUN or EXIT PROGRAM statement, to pass a value to the invoking COBOL or C program or the X/Open environment. It may be referenced subsequent to a call to a COBOL program to obtain the RETURN-CODE set by that program, or subsequent to a call to a C program to obtain the value returned.

8.10 EXTERNAL NAMES AND CASE CONVENTIONS

In portable programs the following restrictions apply to the names that have external scope:

- program-name in the PROGRAM-ID paragraph and end program header (upper-case letters and digits, no hyphens and no leading digits)
- file-name in the SELECT and FD clauses (upper-case letters and digits)
- working storage items having the EXTERNAL clause (lower-case letters, digits and hyphens; hyphens are converted to underscores)

When a character is specified that is not mentioned above, the rules are as specified by Standard COBOL or, if not specified by Standard COBOL, implementation-dependent.

8.11 COPY LIBRARY LOCATION

The constraints on the location of the COPY library members differ from compiler to compiler. Application developers should assume that the only way of defining such a location is by the definition of a path-name-literal relative to the current directory or the system **/** (root) directory. The ability to define the name of a directory in the ''OF library-name'' clause should not be assumed.

Appendix A

Future Directions

A.1 FILE SHARING AND RECORD LOCKING

Currently, record locks can only be acquired via syntax on the READ statement. Some COBOL implementations allow the WRITE and REWRITE statements to acquire record locks, and this is considered a useful feature by X/Open. X/Open hopes to be able to include syntax to allow this capability in the future.

When a (sequential) READ, without explicit or implicit lock, encounters a locked record, it is currently implementation-dependent whether the statement is successful. X/Open hopes to resolve this issue in the future by defining either unsuccessful or successful completion of the statement.

A.2 FREE FORM REFERENCE FORMAT

X/Open expects the restriction on disallowing both reference formats (fixed and free form) in the same compilation unit will be removed in the future. X/Open expects that the mechanism for selection of reference format will be a compiler directive, currently under discussion at the CODASYL COBOL Committee.

A.3 INTERNATIONALISATION

X/Open expects the restriction on the use of Standard COBOL syntax mentioned in **Section 6.2**, **Single-byte Internationalisation Support** will be removed in the future.

X/Open expects the restriction on the use of variable length records, mentioned in **Section 6.2**, **Single-byte Internationalisation Support** will be removed in the future.

The method of specifying that a COBOL program is to use the internationalisation facilities is currently implementation-dependent. X/Open expects COBOL syntax for this to be defined in the future.

The method of specifying that a COBOL indexed, sort or merge file is to use the collating sequence defined for the selected language is currently implementation-dependent. X/Open expects this to be defined in the future.

Accessing a file in another language setting than was current when the file was created, is currently not possible. X/Open expects that this restriction will be removed in the future.

Specification of the language within indexed files, if possible, is currently implementation-dependent. X/Open expects this to be defined in the future.

Currently, a multi-byte character is not equivalent to any single-byte character. In the future, X/Open COBOL may define for COBOL words the equivalence of single-byte letters to their corresponding multi-byte letters and the equivalence of multi-byte lower-case letters to their corresponding upper-case letters.

Culturally-sensitive locale settings are currently not defined for national-character support. X/Open expects this restriction to be removed in the future. This will define the following aspects discussed in **Section 6.3**, **National-character Support**:

- HIGH-VALUE and LOW-VALUE
- ASCENDING KEY and DESCENDING KEY phrase in the OCCURS clause for use in the SEARCH...ALL statement
- RECORD KEY and ALTERNATE RECORD KEY items when referenced in START or READ NEXT statements
- comparisons between national-character data items

Index