VSI OpenVMS
VSI COBOL Reference Guide

Document Number: XX-XXXXXX-XXX

Publication Date: month year

This manual provides reference information and syntax for the VSI COBOL programming language on its platforms: OpenVMS Alpha, OpenVMS Industry Standard 64, and Tru64 UNIX Alpha.

Revision Update Information: This is a new manual.

Operating system and Version: VSI OpenVMS Version X.X
Software Version: VSI COBOL Version X.X
Legal Notice

Confidential computer software. Valid license from VSI required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

The information contained herein is subject to change without notice. The only warranties for VSI products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. VSI shall not be liable for technical or editorial errors or omissions contained herein.

HPE, HPE Integrity, HPE Alpha, and HPE Proliant are trademarks or registered trademarks of Hewlett Packard Enterprise.

Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Oracle is a registered trademark of Oracle and/or its affiliates.

The VSI OpenVMS documentation set is available on DVD.
Appendix D. Report Writer Presentation Rules and Tables ......................................... 489
Appendix C. File Status Values ....................................................................................... 485
Appendix B. Character Sets  ............................................................................................. 477
Appendix A. VSI COBOL Reserved Words ................................................................... 461
Chapter 8. Source Text Manipulation ............................................................................. 441
  8.1. Text-Word Definition Rules ........................................................................................... 441
Appendix A. VSI COBOL Reserved Words ................................................................. 461
Appendix B. Character Sets ............................................................................................ 477
Appendix C. File Status Values ..................................................................................... 485
Appendix D. Report Writer Presentation Rules and Tables ........................................ 489
  D.1. Organization ........................................................................................................ 489
  D.2. LINE NUMBER Clause Notation ......................................................................... 489
  D.3. LINE NUMBER Clause Sequence Substitutions ................................................. 490
  D.4. Saved-Next-Group-Integer Description ............................................................ 490
  D.5. REPORT HEADING Group Presentation Rules .................................................... 490
  D.6. PAGE HEADING Group Presentation Rules ....................................................... 491

Chapter 8. Source Text Manipulation ............................................................................. 441
  8.1. Text-Word Definition Rules ........................................................................................... 441

VSI COBOL Reference Guide
D.7. Body Group Presentation Rules ................................................................. 492
D.8. PAGE FOOTING Group Presentation Rules ..................................................... 496
D.9. REPORT FOOTING Group Presentation Rules ................................................. 497
Appendix E. RTL Routines for Accessing the RAB and FAB Structures (OpenVMS
Alpha and I64 Only) ...................................................................................... 499
Index ........................................................................................................ 503
Preface

This book describes the constructs and rules of the VSI COBOL programming language, which is a VSI Company implementation of COBOL (COmmon Business-Oriented Language) for the OpenVMS and Tru64 UNIX platforms. It includes information about language syntax and semantics, as well as information about adherence and extensions to various COBOL standards.

This documentation set also includes the VSI COBOL User Manual and, optionally, the HPE COBOL DBMS Database Programming.

VSI COBOL is the new name for what has formerly been known as HP COBOL, Compaq COBOL, DEC COBOL, DIGITAL COBOL. VSI COBOL, unmodified, refers to the following products:

VSI COBOL for OpenVMS Industry Standard 64
VSI COBOL for OpenVMS Alpha
VSI COBOL for Tru64 UNIX

Any references to the former names in product documentation or other components should be construed as references to the VSI COBOL names.

1. About VSI

VMS Software, Inc., (VSI) is an independent software company licensed by Hewlett Packard Enterprise to develop and support the OpenVMS operating system.

VSI seeks to continue the legendary development prowess and customer-first priorities that are so closely associated with the OpenVMS operating system and its original author, Digital Equipment Corporation.

2. Intended Audience

This manual is intended for experienced applications programmers who have a thorough understanding of the COBOL language and some familiarity with their operating system. This is not a tutorial manual.

If you are a new COBOL user, you may need to read introductory COBOL textbooks or take COBOL courses.

3. Structure of This Document

This manual is organized as follows:

- Chapter 1, Overview of the COBOL Language presents the elements of the COBOL language, describes two format options for a COBOL program, and explains how the remaining chapters organize and present the COBOL general formats.

- Chapter 2, Organization of a COBOL Program describes the organization of a COBOL program. It presents the general format for the four COBOL divisions and introduces the concept of contained programs. This chapter shows the relationship between a program name and a source file name.

- Chapter 3, Identification Division describes the general format and contents of the Identification Division. It explains how to identify a COBOL program and its source listing.

- Chapter 4, Environment Division describes the general format and contents of the Environment Division. It explains how to describe the program's physical environment.

- Chapter 5, Data Division describes the general format and contents of the Data Division. It explains how to describe data the program receives, creates, manipulates, and produces as output.
• Chapter 6, Procedure Division describes the general format and contents of the Procedure Division. It describes COBOL verbs, which process the files and data in the Environment and Data Divisions.

• Chapter 7, Intrinsic Functions describes the general format and use of the intrinsic functions.

• Chapter 8, Source Text Manipulation describes the general format of the COPY and REPLACE statements.

• Appendix A, VSI COBOL Reserved Words lists the VSI COBOL reserved words, which are words that cannot be used as system names or user-defined names.

• Appendix B, Character Sets lists the ASCII, EBCDIC, and NATIVE character sets.

• Appendix C, File Status Values lists the exception condition values that can appear in File Status data items.

• Appendix D, Report Writer Presentation Rules and Tables contains individual presentation rules and tables for each type of report group.

• Appendix E, RTL Routines for Accessing the RAB and FAB Structures (OpenVMS Alpha and I64 Only) describes RTL routines for accessing the RAB and FAB structures on OpenVMS systems.

• The Index indexes and references terms and concepts in this manual.

4. Associated Documents

The following documents contain additional information directly related to various topics covered in this manual:

VSI COBOL User Manual

This manual describes how to use features of the VSI COBOL language to develop programs on the Tru64 UNIX or the OpenVMS operating systems on Alpha, I64, and VAX.

Release Notes

Consult the VSI COBOL release notes for your installed version for late corrections and new features.

On the OpenVMS Alpha, I64 operating system, the release notes are in:

SYSSHELP:COBOL nnn.RELEASE_NOTES (ASCII text)
SYSSHELP:COBOL nnn.RELEASE_NOTES.PS

Where nnn is the version and release number.

On the Tru64 UNIX, the release notes are in:

/usr/lib/cmplrs/cobol/relnotes

DEC COBOL Installation Guide for VSI UNIX Systems

This manual provides instructions for installing VSI COBOL on the Tru64 UNIX.

DEC COBOL Installation Guide for OpenVMS Alpha Systems

This manual provides instructions for installing VSI COBOL on the OpenVMS Alpha and OpenVMS I64 operating systems.

HPE COBOL DBMS Database Programming

This manual provides information on using VSI COBOL for database programming with Oracle CODASYL DBMS on the OpenVMS Alpha, the OpenVMS I64, or OpenVMS VAX operating systems.
The OpenVMS Calling Standard and other manuals in the OpenVMS Documentation Set

This set contains information about using the features of the OpenVMS I64, OpenVMS Alpha operating systems and their tools.

The Tru64 UNIX Documentation Set

This set contains introductory and detailed information about using the features of the Tru64 UNIX operating system and its tools.

The Alpha Architecture Reference Manual

This manual is available from Digital Press.

5. Related Documents

For additional information about VSI OpenVMS products and services, visit:

https://www.vmssoftware.com

6. Conventions

The following product names may appear in this manual:

• VSI OpenVMS for Integrity servers
• OpenVMS I64
• I64

All three names — the longer form and the two abbreviated forms — refer to the version of the OpenVMS operating system that runs on the Intel® Itanium® architecture.

The following typographic conventions may be used in this manual:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECORD KEY IS</td>
<td>Underlined uppercase words are required when used in a general format.</td>
</tr>
<tr>
<td></td>
<td>Uppercase words not underlined are optional.</td>
</tr>
<tr>
<td>sortfile</td>
<td>Lowercase words used in a general format are generic terms that indicate entries you must provide.</td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{ SEQUENTIAL</td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{ RECORD</td>
</tr>
<tr>
<td>{ {</td>
<td></td>
</tr>
<tr>
<td>Convention</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>{ { COMMON</td>
<td>INITIAL } }</td>
</tr>
<tr>
<td>.</td>
<td>A vertical ellipsis indicates that not all of the statements are shown.</td>
</tr>
<tr>
<td>Format</td>
<td>Program examples are shown in terminal format, rather than in ANSI standard format.</td>
</tr>
<tr>
<td>special-character words</td>
<td>The following symbols, when used in a general format, constitute required special-character words:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Plus sign (+)</td>
<td></td>
</tr>
<tr>
<td>Minus sign (-)</td>
<td></td>
</tr>
<tr>
<td>Single (=) and double (==) equal signs</td>
<td></td>
</tr>
<tr>
<td>Less than (&lt;) or greater than (&gt;) symbols</td>
<td></td>
</tr>
<tr>
<td>Less than or equal to (&lt;=) and greater than or equal to (&gt;=) symbols</td>
<td></td>
</tr>
<tr>
<td>Period (.)</td>
<td></td>
</tr>
<tr>
<td>Colon (:).</td>
<td></td>
</tr>
<tr>
<td>Single (<em>) and double (</em>**) asterisks</td>
<td></td>
</tr>
<tr>
<td>Slash (/)</td>
<td></td>
</tr>
<tr>
<td>Left parenthesis ( bold or right parenthesis (bold))</td>
<td></td>
</tr>
<tr>
<td>quotation mark</td>
<td>The term quotation mark is used to refer to the double quotation mark character (&quot;).</td>
</tr>
<tr>
<td>apostrophe</td>
<td>The term apostrophe is used to refer to the single quotation mark character (').</td>
</tr>
<tr>
<td>user input</td>
<td>In examples, user input (what you enter) is shown as monospaced text.</td>
</tr>
<tr>
<td>light blue color</td>
<td>Light blue color indicates the language extensions to the Fortran 95 Standard.</td>
</tr>
<tr>
<td>report file</td>
<td>Bold type indicates a new term.</td>
</tr>
<tr>
<td>full-file-name</td>
<td>This syntax term refers to the name of a file and the device and directory, or path, in which it is located. For example:</td>
</tr>
</tbody>
</table>
|                                    | DISK2$: [HOME.PUBLIC]FILENAME.TXT; (OpenVMS file specification)  /
disk2/home/public/filename.txt (Tru64 UNIX file specification) |
| compiler option                    | This term refers to command-line qualifiers (OpenVMS Alpha and I64 systems) or flags (Tru64 UNIX systems). For example:             |
|                                    | /LIST (OpenVMS qualifier specification) -list (Tru64 UNIX flag specification)                                                         |
| COBOL                               | This term refers to language information common to ANSI-85 COBOL, VSI COBOL, and VSI COBOL.                                             |
7. References

The following table shows certain references and their respective meanings in this manual:

<table>
<thead>
<tr>
<th>Reference</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha</td>
<td>OpenVMS Alpha or Tru64 UNIX Alpha operating system</td>
</tr>
<tr>
<td>OpenVMS</td>
<td>OpenVMS Alpha or OpenVMS I64 operating system</td>
</tr>
<tr>
<td>Tru64 UNIX</td>
<td>Tru64 UNIX Alpha operating system</td>
</tr>
</tbody>
</table>

Tru64 UNIX was formerly known as DEC OSF/1 or as DIGITAL UNIX. VSI COBOL was formerly known as HP COBOL, Compaq COBOL, DIGITAL COBOL, or DEC COBOL.

8. How to Order Additional Documentation

For information about how to order additional documentation, email the VSI OpenVMS information account: <info@vmssoftware.com>. We will be posting links to documentation on our corporate website soon.

9. VSI Encourages Your Comments

You may send comments or suggestions regarding this manual or any VSI document by sending electronic mail to the following Internet address: <docinfo@vmssoftware.com>. Users who have OpenVMS support contracts through VSI can contact <support@vmssoftware.com> for help with this product. Users who have OpenVMS support contracts through HPE should contact their HPE Support channel for assistance.
Chapter 1. Overview of the COBOL Language

This chapter provides information about the structure and language of COBOL source programs. It describes the elements of the COBOL language, reference formats, and language organization.

The COBOL language consists of the following components:

- Programs
- Divisions
- Sections
- Paragraphs
- Sentences
- Statements
- Clauses
- Entries
- Words
- Characters

A separately compiled COBOL program is a program that, together with its contained programs (if present), is compiled separately from all other programs. Each COBOL program is divided into four parts, called divisions: the Identification Division, Environment Division, Data Division, and Procedure Division. Divisions can contain sections, which in turn can contain paragraphs. Paragraphs can contain sentences, clauses, statements, or entries.

The building blocks of these language components include the COBOL character set, character-strings, separators, punctuation, and literals.

A COBOL program is a string of characters that is syntactically correct according to the COBOL language rules.

1.1. The COBOL Character Set

The COBOL character set, shown in Table 1.1, “The COBOL Character Set”, is used to form character-strings and separators.

The only components of a COBOL program that can contain characters outside this set are nonnumeric literals, comment-entries, and comment lines. Appendix B, Character Sets specifies the more inclusive computer character sets these elements can use.

### Table 1.1. The COBOL Character Set

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1, ..., 9</td>
<td>digit</td>
</tr>
<tr>
<td>A, B, ..., Z</td>
<td>letter</td>
</tr>
<tr>
<td>a, b, ..., z</td>
<td>lowercase letter (equivalent to letter)</td>
</tr>
<tr>
<td>+</td>
<td>plus sign</td>
</tr>
<tr>
<td>-</td>
<td>minus sign (hyphen)</td>
</tr>
<tr>
<td>Character</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------</td>
</tr>
<tr>
<td>*</td>
<td>asterisk</td>
</tr>
<tr>
<td>/</td>
<td>slash (stroke, virgule)</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
</tr>
<tr>
<td>=</td>
<td>equal sign</td>
</tr>
<tr>
<td>$</td>
<td>currency sign</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than symbol</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than symbol</td>
</tr>
<tr>
<td>:</td>
<td>colon</td>
</tr>
<tr>
<td>_</td>
<td>underline (underscore)</td>
</tr>
<tr>
<td></td>
<td>space</td>
</tr>
<tr>
<td>Tab</td>
<td>horizontal tab</td>
</tr>
<tr>
<td>(</td>
<td>left parenthesis</td>
</tr>
<tr>
<td>)</td>
<td>right parenthesis</td>
</tr>
<tr>
<td>.</td>
<td>comma (decimal point)</td>
</tr>
<tr>
<td>;</td>
<td>semicolon</td>
</tr>
<tr>
<td>.</td>
<td>period (decimal point, full stop)</td>
</tr>
<tr>
<td>&quot;</td>
<td>quotation mark (double quotation mark)</td>
</tr>
<tr>
<td>'</td>
<td>apostrophe (single quotation mark)</td>
</tr>
<tr>
<td>{</td>
<td>left brace</td>
</tr>
<tr>
<td>}</td>
<td>right brace</td>
</tr>
<tr>
<td>[</td>
<td>left bracket</td>
</tr>
<tr>
<td>]</td>
<td>right bracket</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>double left-angle brackets</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>double right-angle brackets</td>
</tr>
</tbody>
</table>

Except in nonnumeric literals, the compiler treats lowercase letters as if they were uppercase. Therefore, a program can contain COBOL words without regard to case. For example, the compiler recognizes the COBOL words in each of the following pairs as identical:

<table>
<thead>
<tr>
<th>WORKING-STORAGE</th>
<th>Working-Storage</th>
<th>FILE-A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input file-a</td>
<td>FILE-A</td>
</tr>
<tr>
<td>INSPECT</td>
<td>InSpect</td>
<td></td>
</tr>
</tbody>
</table>

### 1.2. Character Strings

A **character-string** is a character or a sequence of contiguous characters that form a COBOL word, a literal, a PICTURE character-string, or a comment-entry. Separators delimit character-strings. The following sections describe these topics in detail.

#### 1.2.1. COBOL Words

A **COBOL word** is a character-string of not more than 31 characters that forms one of the following:

- A user-defined word
- A system-name
A user-defined word or system-name cannot be a reserved word. However, a program can use the same COBOL word as both a user-defined word and a system-name. The compiler determines the word's class from its context.

### 1.2.1.1. User-Defined Words

A user-defined word is a COBOL word that you must supply to satisfy the format of a clause or statement. This word consists of characters selected from the set A to Z, 0 to 9, the currency sign ($), underline (_), and hyphen (-). Throughout this manual, and except where specific rules apply, the hyphen (-) and the underline (_) are treated as the same character in a user-defined word. The underline (_) however, can begin or end a user-defined word, and the hyphen (-) cannot. By convention, names containing a currency sign ($) are reserved for VSI.

Table 1.2, “COBOL User-Defined Words” provides brief descriptions of the COBOL user-defined words.

**Table 1.2. COBOL User-Defined Words**

<table>
<thead>
<tr>
<th>User-Defined Word</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabet-Name</td>
<td>Assigns a name to a character set, collating sequence, or both. Alphabet-names must be defined in the SPECIAL-NAMES paragraph. (See SPECIAL-NAMES in Chapter 4: Environment Division.)</td>
</tr>
<tr>
<td>Class-Name</td>
<td>Relates a name to a specified set of characters listed in that clause. (See SPECIAL-NAMES in Chapter 4: Environment Division.)</td>
</tr>
<tr>
<td>Condition-Name</td>
<td>Assigns a name to a value, set of values, or range of values in the complete set of values that a data item can have. Data items with one or more associated condition-names are called conditional variables.</td>
</tr>
<tr>
<td></td>
<td>Data Division entries define condition-names. Names assigned in the SPECIAL-NAMES paragraph to the &quot;on&quot; or &quot;off&quot; status of switches are also condition-names.</td>
</tr>
<tr>
<td>Data-Name</td>
<td>Names a data item described in a data description entry. When specified in a general format, data-name cannot be reference modified, subscripted, indexed, or qualified unless specifically allowed by the rules for that format.</td>
</tr>
<tr>
<td>File-Name</td>
<td>Names a file connector. A file connector is a storage area that contains information about a file and is the link between:</td>
</tr>
<tr>
<td></td>
<td>• A file-name and a physical file</td>
</tr>
<tr>
<td></td>
<td>• A file-name and its associated storage area</td>
</tr>
<tr>
<td></td>
<td>File description entries and sort-merge file description entries describe file connectors.</td>
</tr>
<tr>
<td>Index-Name</td>
<td>Names an index associated with a specific table.</td>
</tr>
<tr>
<td>Level-Number</td>
<td>Is a one- or two-digit number that describes a data item's special properties or its position in the structure of a record. (See Sections 5.1.1 and 5.1.2.)</td>
</tr>
</tbody>
</table>
### User-Defined Word

<table>
<thead>
<tr>
<th>User-Defined Word</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Library-Name</strong></td>
<td>Names a COBOL library used in a source program compilation. (See the COPY statement in Chapter 8, Source Text Manipulation.)</td>
</tr>
<tr>
<td><strong>Mnemonic-Name</strong></td>
<td>Associates a name with a system-name, such as CONSOLE, SYSERR, ARGUMENT-NUMBER, ENVIRONMENT-NAME, C01, OR SWITCH-8. (See SPECIAL-NAMES in Chapter 4, Environment Division.)</td>
</tr>
<tr>
<td><strong>Paragraph-Name</strong></td>
<td>Names a Procedure Division paragraph. (See the section called “Paragraph, Paragraph Header, Paragraph-Name”.) Paragraph-names are equivalent only if they are identical; that is, if they are composed of the same sequence and number of digits and/or characters. For example:</td>
</tr>
<tr>
<td><strong>Program-Name</strong></td>
<td>Identifies a COBOL source program. (See the PROGRAM-ID paragraph in Chapter 3, Identification Division, and the section on CALL in Chapter 6, Procedure Division, for a description of case-sensitivity on the Tru64 UNIX. Also refer to the VSI COBOL User Manual for a description of the -names lowercase, -names uppercase, and -names as_is flags.)</td>
</tr>
<tr>
<td><strong>Record-Name</strong></td>
<td>Names a data item described with level-number 01 or 77.</td>
</tr>
<tr>
<td><strong>Report-Name</strong></td>
<td>Names a report produced by the Report Writer Control System (RWCS). (See the REPORT clause in Chapter 5, Data Division.)</td>
</tr>
<tr>
<td><strong>Screen-Name (Alpha, I64)</strong></td>
<td>Names a screen item defined in the SCREEN SECTION of a program. (See the Screen Description (Alpha, I64) section of Chapter 5, Data Division.)</td>
</tr>
<tr>
<td><strong>Section-Name</strong></td>
<td>Names a Procedure Division section. Section-names are equivalent only if they are identical; that is, when they are composed of the same sequence and number of digits and/or characters. (See the section called “Section Header”.)</td>
</tr>
<tr>
<td><strong>Segmented-Key-Name</strong></td>
<td>Identifies a segmented key, which is a concatenation of one or more (up to eight) data items (segments) within a record associated with an indexed file. A segmented key is a form of primary or alternate key. It offers flexibility in defining record description entries for</td>
</tr>
</tbody>
</table>
### User-Defined Word | Purpose
--- | ---
Indexed Files | indexed files. (Refer to the section on segmented keys in the VSI COBOL User Manual.)
Segment-Number | Is a 1- or 2-digit number that classifies a Procedure Division section for segmentation. In VSI COBOL programs, segment-numbers specify independent and fixed segments. (See Section 6.7, “Segmentation”.)
Symbolic-Character | Identifies a user-defined figurative constant.
Text-Name | Identifies library text in a COBOL library. (See the COPY statement in Chapter 8, Source Text Manipulation.)

Within a given program, but excluding any contained program, the user-defined words are grouped into the following disjoint sets:

- alphabet-names
- class-names
- condition-names, data-names, and record-names
- file-names
- index-names
- library-names
- mnemonic-names
- paragraph-names
- program-names
- report-names
- screen-names
- section-names
- segmented-key-names
- symbolic-characters
- text-names

All user-defined words in a program, except segment-numbers and level-numbers, can belong to only one of these sets. User-defined words in each set must be unique, except as described in the rules for uniqueness of reference. (See Section 6.2, “Uniqueness of Reference”).

Except for section-names, paragraph-names, segment numbers, and level-numbers, all user-defined words must contain at least one alphabetic character. Segment-numbers and level-numbers need not be unique. Any segment-number or level-number can be the same as any other segment-number or level-number.

### 1.2.1.2. System-Names

System-names are COBOL words that refer to the program's operating environment. The same COBOL word can be used in a program as both a user-defined word and a system-name. The compiler determines the word's class from its context.

The system-names are as follows:

- ALPHA
- ASCII
- CARD-READER
- CONSOLE
- CONTIGUOUS
- CONTIGUOUS-BEST-TRY
- C01
1.2.1.3. Reserved Words

A reserved word can be used only as specified in the general formats. It cannot be a user-defined word. (See Appendix A, VSI COBOL Reserved Words for a list of reserved words.)

The three types of reserved words follow:

• Required words
• Optional words
• Special-purpose words

Required Word

A required word must be used when its format is used in a program.

The two types of required words are keywords and special character words. In general formats, keywords are uppercase and underlined. Arithmetic operators and relation characters are special character words; they are not underlined in the general format.

In the following sample format, the keywords are COMPUTE, ROUNDED, SIZE, ERROR, NOT, and END-COMPUTE. The equal sign (=) is a special-character word.

```
COMPUTE | (result | ROUNDED | ... = arithmetic expression
| ON SIZE ERROR string1 |
| NOT ON SIZE ERROR string2 |
| END-COMPUTE |
```

Optional Words

In general formats, uppercase words that are not underlined are optional words. They can make a program more human-readable, but have no semantic effect. In the previous sample format, ON is an optional word.

Special-Purpose Words

The two types of special-purpose words are figurative constants and special registers. Figurative constants name and refer to specific constant values and are described in detail in Section 1.2.3, “Figurative Constants”. Special registers name and refer to special storage areas that the compiler provides.

The VSI COBOL special registers are primarily used to store information related to or produced by specific VSI COBOL features. Table 1.3, “Special Registers” shows the special registers, their usage, and their descriptions.
### Table 1.3. Special Registers

<table>
<thead>
<tr>
<th>Special Register</th>
<th>Usage—Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETURN-CODE (Alpha, I64)</td>
<td>X/Open—Names an VSI COBOL special register that may be used to set a return value for a calling program or to retrieve the value returned from a called program. It is represented by PIC S9(9) USAGE IS COMP. It is implicitly defined with GLOBAL scope. The RETURN-CODE register is initialized with the platform-specific success code. On OpenVMS Alpha and OpenVMS I64, it is initialized to one. On Tru64 UNIX it is initialized to zero. The RETURN-CODE special register can be set by a called program, prior to the execution of a STOP RUN or EXIT PROGRAM statement, to pass a value to the calling program or the execution environment. For a calling program, it can be read, subsequent to the CALL, to obtain the value of the RETURN-CODE set by the called program. On Tru64 UNIX the main program sets the shell variable status to the value of the RETURN-CODE. On OpenVMS Alpha and OpenVMS I64 the main program sets the symbol $STATUS to the value of the RETURN-CODE. If you use the GIVING phrase on the CALL statement or on the Procedure Division header, specifying a data item as its argument, this data item (instead of RETURN-CODE) receives the return value. Note that you can specify the special register RETURN-CODE as the argument to GIVING, in which case RETURN-CODE receives the return value. For more information on the relationship between the GIVING phrase and the RETURN-CODE special register, see Table 6.7, “Relation of GIVING Phrase to RETURN-CODE Special Register (Alpha, I64)” in Chapter 6, Procedure Division. Because the reserved word RETURN-CODE is one of the X/Open reserved words, you cannot use the nokopen keyword in the reserved_words compiler option if you want to use the RETURN-CODE special register. For related information, see Section 6.8, “General Formats and Rules for Statements” for the syntax and description of the GIVING phrase of the Procedure Division header; and the CALL statement for the syntax and description of CALL GIVING.</td>
</tr>
<tr>
<td>LINAGE-COUNTER</td>
<td>LINAGE files—A line counter that the compiler provides when a file description entry contains a LINAGE clause. Its value is the number of the current record within the page body. (See the LINAGE clause in Chapter 5, Data Division.) The implicit</td>
</tr>
<tr>
<td>Special Register</td>
<td>Usage—Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LINAGE-COUNTER</td>
<td>Size of LINAGE-COUNTER is nine decimal digits represented by PIC S9(9) COMP. You can qualify LINAGE-COUNTER with a file-name. Procedure Division statements and the SOURCE clause of the Report Section can access the value of LINAGE-COUNTER but cannot change its value. LINAGE-COUNTER is global if file-name is global and external if file-name is external.</td>
</tr>
<tr>
<td>PAGE-COUNTER</td>
<td>REPORT WRITER—A page counter that the compiler provides for each report in the Report Section of the Data Division. You can qualify PAGE-COUNTER with a report-name. Its value is the number of the current page within a report. The implicit size of PAGE-COUNTER is six unsigned decimal digits represented by PIC 9(6) COMP. The Report Writer Control System (RWCS) maintains the value of PAGE-COUNTER and uses this value to number the pages of a report. The SOURCE clause of the Report Section can reference PAGE-COUNTER. The values in PAGE-COUNTER range from 1 to 999999 and can be altered by Procedure Division statements.</td>
</tr>
<tr>
<td>LINE-COUNTER</td>
<td>REPORT WRITER—A line counter that the compiler generates for each report in the Report Section of the Data Division. It may be qualified by a report-name. Its value is the number of the current line within a page. (See PAGE-COUNTER.) The implicit size of LINE-COUNTER is six unsigned decimal digits represented by PIC 9(6) COMP. The Report Writer Control System (RWCS) maintains the value of LINE-COUNTER and uses this value to determine the vertical positioning of a report. The SOURCE clause of the Report Section can reference LINE-COUNTER. The values in LINE-COUNTER range from 0 to 999999. Procedure Division statements can access the values in LINE-COUNTER; however, only the RWCS can change its value.</td>
</tr>
<tr>
<td>RMS-STS <em>(OpenVMS)</em></td>
<td>RMS—Contains the primary RMS status value of an I/O operation. (RMS-STV contains the secondary value.) RMS-STS provides additional information on COBOL File Status values resulting from I/O operations. It is represented by PIC S9(9) USAGE IS COMP. You must qualify RMS-STS with a file-name. If the file-name is global, RMS-STS is also global. If the file-name is external, RMS-STS is also external. Before the program opens the file for the first time, the value of RMS-STS is undefined. After your program executes an OPEN or CLOSE statement, RMS-STS is set to the value of the STS field in the associated file access block (FAB). After executing a READ, WRITE, REWRITE, DELETE, START, or UNLOCK statement, RMS-STS is set to the value of the STS field in the associated record access block (RAB).</td>
</tr>
</tbody>
</table>
### Special Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Usage—Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMS-STV 1(OpenVMS)</td>
<td>RMS—Contains the secondary (RMS-STS is primary) RMS status value of an I/O operation. The interpretation of this value is dependent on the value in RMS-STS. It is represented by PIC S9(9) USAGE IS COMP. You must qualify RMS-STV with a file-name. If the file-name is global, RMS-STV is also global. If the file-name is external, RMS-STV is also external. The value in RMS-STV is undefined prior to the initial OPEN of the file. After your program executes an OPEN or CLOSE statement, RMS-STV is set to the value of the STV field in the associated FAB. After executing a READ, WRITE, REWRITE, DELETE, or START statement, RMS-STV is set to the value of the STV field in the associated RAB.</td>
</tr>
<tr>
<td>RMS-Filename 1(OpenVMS)</td>
<td>RMS—Names the complete RMS filename. It consists of 255 alphanumeric characters represented by PIC X(255) USAGE IS DISPLAY. You must qualify it with a file-name. If the file-name is global, RMS-Filename is also global. If the file-name is external, RMS-Filename is also external. Before the program opens the file for the first time, the value of RMS-Filename is undefined. For each COBOL OPEN statement, RMS-Filename is set to the complete RMS file specification string of filename: for example, DB21:[COBOL]MASTER.DAT.</td>
</tr>
<tr>
<td>RMS-CURRENT-STS 1(OpenVMS)</td>
<td>RMS—Names an VSI COBOL exception condition register. It contains the primary RMS status value of the most recent RMS I/O operation, regardless of the file operated on. (RMS-CURRENT-STV contains the secondary value.) It is represented by PIC S9(9) USAGE IS COMP. Since this register can contain the primary RMS status value for any file, you must not qualify it with a file-name. After your program executes any RMS I/O operation, it sets RMS-CURRENT-STS to the value contained in RMS-STS for that file.</td>
</tr>
<tr>
<td>RMS-CURRENT-STV 1(OpenVMS)</td>
<td>RMS—Names an VSI COBOL exception condition register. It contains the secondary RMS status value of the most recent RMS I/O operation, regardless of the file operated on. (RMS-CURRENT-STS contains the primary value.) It is represented by PIC S9(9) USAGE IS COMP. Since this register can contain the secondary RMS status value for any file, you must not qualify it with a file-name. After your program executes any RMS I/O operation, it sets RMS-CURRENT-STV to the value contained in RMS-STV for that file.</td>
</tr>
<tr>
<td>RMS-CURRENT-Filename 1(OpenVMS)</td>
<td>RMS—Names an VSI COBOL exception condition register. It contains the complete RMS file specification string of the file most recently operated on by an I/O statement. It consists of 255</td>
</tr>
</tbody>
</table>
### 1.2.1.4. Function-Names

A *function-name* is the name of a function as shown in Table 7.1: Intrinsic Functions. Note that function-names are not reserved words and may appear in a different context in a program as a user-defined word or a system-name.

### 1.2.2. Literals

A *literal* is a character-string whose value is specified by: (1) the ordered set of characters it contains, or (2) a reserved word that is a figurative constant.

VSI COBOL provides two types of literals: numeric and nonnumeric. Numeric literals include floating-point literals and nonnumeric literals include hexadecimal and national literals. Floating-point, hexadecimal, and national literals are VSI extensions. The following two sections describe literals in detail.

#### 1.2.2.1. Numeric Literals

A *numeric literal* is a character string of 1 to 33 characters on Alpha and I64 selected from the digits 0 to 9, the plus sign (+), the minus sign (-), and the decimal point (.).

The value of a numeric literal is the algebraic quantity represented by the characters in the literal.

**Syntax Rules**

1. A numeric literal must contain at least 1 digit and not more than 31 digits on Alpha and I64.
2. A numeric literal must not contain more than one sign character, which must be the leftmost character. If the literal is unsigned, its value is positive.
3. A numeric literal must not contain more than one decimal point. The decimal point is treated as an assumed decimal point. It can be used anywhere in the literal except as the rightmost character.
   
   If a numeric literal contains no decimal point, it is an integer.
4. The compiler treats a numeric literal enclosed in quotation marks as a nonnumeric literal.

Table 1.4, “Numeric Literals” provides examples of numeric literals.

<table>
<thead>
<tr>
<th>Literal</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

---

1. Procedure Division statements can the values or strings stored in the RMS special registers; however, only the RMS facility can the contents of the registers. Refer to the VSI COBOL User Manual for programming examples. For an explanation and a listing of RMS STS and STV values, refer to the OpenVMS System Messages and Recovery Procedures Reference Manual, an archived manual available on the OpenVMS Documentation CD-ROM, or the online OpenVMS Help Message utility. Refer to the OpenVMS Record Management Services Reference Manual for information on RMS. (RMS is on OpenVMS systems only.)

2. The FILE STATUS data item (see Section 6.6.8: I-O Status) provides the primary source of status information for the file I-O verbs, and RMS-STS and RMS-STV provide supplementary information.
Floating-Point Literals

A floating-point literal, a VSI extension to numeric literals, is a character-string whose value is specified by 4 to 37 characters on Alpha and I64, selected from the digits 0 to 9, the plus sign (+), the minus sign (-), the decimal point (.), and the letter E (uppercase or lowercase).

You can use floating-point literals to achieve a wider range of numeric literal values.

Syntax Rules

1. A floating-point literal must be between 4 and 37 (Alpha, I64) characters in length.
2. A floating-point literal must contain the following characters:
   • At least 1 digit to the left of the E
   • A decimal point to the left of the E
   • An E (uppercase or lowercase)
   • At least 1 digit to the right of the E
3. The maximum number of characters to the left of the E is 33 (Alpha, I64) of which no more than 31 can be digits.
4. The maximum number of characters to the right of the E is 4 (Alpha, I64) of which no more than 3 can be digits.
5. A floating-point literal must not contain more than two sign characters as follows:
   • The first character of the literal
   • The first character following the E
6. If the first character of the literal is not a sign character, the literal is positive.
7. If the first character following the E is not a sign character, the value of the numeric component following the E is positive.
8. A floating-point literal must contain only one decimal point that can appear only to the left of the E.
9. A comma must be used in place of the decimal point, if the DECIMAL POINT IS COMMA clause is specified.

The value of a floating-point literal is the algebraic quantity represented by the characters in the literal that precede the E multiplied by ten raised to the power of the algebraic quantity represented by the characters in the literal following the E.

Table 1.5, “Floating-Point Literals” provides a few examples of floating-point literals.

Table 1.5. Floating-Point Literals

<table>
<thead>
<tr>
<th>Literal</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6e5</td>
<td>160000.0</td>
</tr>
</tbody>
</table>
### 1.2.2.2. Nonnumeric Literals

A nonnumeric literal is a character-string of 0 to 256 characters. It is delimited on both ends by quotation marks (" ) or apostrophes ('). A nonnumeric literal delimited by apostrophes is treated in the same manner as a nonnumeric literal delimited by quotation marks.

The value of a nonnumeric literal is the value of the characters in the character-string. It does not include the quotation marks (or apostrophes) that delimit the character-string. All other punctuation characters in the nonnumeric literal are part of its value.

The compiler truncates nonnumeric literals to a maximum of 256 characters.

#### Syntax Rules

1. A space, left parenthesis, or pseudo-text delimiter (==) must immediately precede the opening quotation mark (or apostrophe).

2. The closing quotation mark (or apostrophe) must be immediately followed by one of the following:
   - Space
   - Comma
   - Semicolon
   - Period
   - Right parenthesis
   - Pseudo-text delimiter

3. If a nonnumeric literal is delimited by quotation marks (" ), two consecutive quotation mark characters in the literal represent one quotation mark character.

4. If a nonnumeric literal is delimited by apostrophes ('), two consecutive apostrophes in the literal represent one apostrophe (').

Table 1.6, “Nonnumeric Literals” provides examples of nonnumeric literals. In these examples, s represents a space character.

#### Table 1.6. Nonnumeric Literals

<table>
<thead>
<tr>
<th>Literal</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ABC &quot;</td>
<td>ABC</td>
</tr>
<tr>
<td>&quot;01 &quot;</td>
<td>01</td>
</tr>
<tr>
<td>&quot;s01 &quot;</td>
<td>s01</td>
</tr>
<tr>
<td>&quot;D &quot; &quot;E &quot; &quot;F &quot;</td>
<td>D &quot;E &quot;F</td>
</tr>
<tr>
<td>&quot;a.b &quot;</td>
<td>a.b</td>
</tr>
<tr>
<td>'GHI '</td>
<td>GHI</td>
</tr>
</tbody>
</table>
Hexadecimal Literals

A hexadecimal literal (a VSI extension to nonnumeric literals) is a character string of 2 to 256 hexadecimal digits. On the left it is delimited by the separator X (or x) immediately followed by a quotation mark (" ) or apostrophe ('); on the right it is delimited by a matching quotation mark or apostrophe. For example:

03 HEX_VAL PIC X VALUE X"00".

The character string consists only of pairs of hexadecimal digits representing a byte value ranging from 00 to FF; hence, only the characters 0 to 9, A to F, and a to f are valid.

The value of a hexadecimal literal is the composite value of the paired hexadecimal representations. The compiler truncates hexadecimal literals to a maximum of 128 hexadecimal representations (pairs of hexadecimal digits).

A hexadecimal literal can be used interchangeably wherever a nonnumeric literal can appear in VSI COBOL syntax. (Thus, hexadecimal literals cannot be used as operands in arithmetic statements.)

Syntax Rules

1. A space, left parenthesis, or pseudo-text delimiter (==) must immediately precede the opening character X (or x).

2. The closing quotation mark or apostrophe must be immediately followed by one of the following:
   - Space
   - Comma
   - Semicolon
Overview of the COBOL Language

- Period
- Right parenthesis
- Pseudo-text delimiter

Table 1.7, “Hexadecimal Literals” provides examples of hexadecimal literals.

Table 1.7. Hexadecimal Literals

<table>
<thead>
<tr>
<th>Literal</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>X &quot;00 &quot;</td>
<td>NUL</td>
</tr>
<tr>
<td>x &quot;0D &quot;</td>
<td>CR</td>
</tr>
<tr>
<td>x &quot;2424 &quot;</td>
<td>$$</td>
</tr>
<tr>
<td>X '7b7a '</td>
<td>{z</td>
</tr>
</tbody>
</table>

National Literals

National literals can be from 0 to 128 2-byte characters (hence 256 bytes). The syntax is:

```
VALUE N"".
```

National literals are made available when /NATIONALITY=JAPAN or -nationality japan is specified.

1.2.3. Figurative Constants

Figurative constants name and refer to specific constant values generated by the compiler. The singular and plural forms of figurative constants are equivalent and interchangeable. Table 1.8, “Figurative Constants” lists the figurative constants.

Table 1.8. Figurative Constants

<table>
<thead>
<tr>
<th>Figurative Constant</th>
<th>Value</th>
</tr>
</thead>
</table>
| ZERO, ZEROS, ZEROES | Represent the value zero, or one or more occurrences of the character 0 from the computer character set, depending on context. In the following example, the first use of the word ZERO represents a zero value; the second represents six 0 characters:

```
03 ABC PIC 9(5) VALUE ZERO.
03 DEF PIC X(6) VALUE ZERO.
```

| SPACE, SPACES | Represent one or more space characters from the computer character set. |
| HIGH-VALUE, HIGH-VALUES | Represent one or more occurrences of the character with the highest ordinal position in the program collating sequence. For example, HIGH-VALUE for the native collating sequence is hexadecimal FF. The value of HIGH-VALUE depends on the collating sequence specified by clauses in the OBJECT-COMPUTER and SPECIAL-NAMES paragraphs. For example, if the program collating sequence is ASCII, |
### Figurative Constant

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGH-VALUE is hexadecimal 7F (hexadecimal FF for EBCDIC). For more information, see OBJECT-COMPUTER and SPECIAL-NAMES sections in Chapter 4: Environment Division.</td>
<td></td>
</tr>
</tbody>
</table>
| LOW-VALUE, LOW-VALUES | Represent one or more occurrences of the character with the lowest ordinal position in the program collating sequence (hexadecimal 00 for the native collating sequence).  
The value of LOW-VALUE depends on the program collating sequence specified by clauses in the OBJECT-COMPUTER and SPECIAL-NAMES paragraphs. For more information, see the OBJECT-COMPUTER and SPECIAL-NAMES sections in Chapter 4: Environment Division. |
| QUOTE, QUOTES | Represent one or more occurrences of the quotation mark character. QUOTE or QUOTES cannot be used in place of a quotation mark to bound a nonnumeric literal. The following examples are not equivalent:  
QUOTE abcd QUOTE  
"abcd" |
| ALL Literal | Represents one or more occurrences of the string of characters making up the literal. The literal must be either nonnumeric, a symbolic-character, or a figurative constant other than ALL literal. For a figurative constant, the word ALL is redundant and serves only to enhance readability.¹ |
| Symbolic-character | Represents one or more occurrences of the character specified as the value of symbolic-character. (See SPECIAL-NAMES in Chapter 4: Environment Division.) |

¹The reserved word ALL, not followed by a literal, can be a subscript of an identifier that is a function argument. (The function must allow a variable number of arguments in this argument position; see Chapter 7, Intrinsic Functions.)

When a figurative constant represents a string of one or more characters, the string's length depends on its context:

- The string's length can vary for a figurative constant in a VALUE IS clause, or for one associated with another data item (for example, when the figurative constant is moved to or compared with another data item). Proceeding from left to right, the compiler repeats the string of characters that represents the figurative constant. It repeats them, character by character, until the size of the resultant string equals that of the associated data item. This is done before and independent of the application of any JUSTIFIED clause specified for the data item.

- When a figurative constant is not associated with another data item (for example, when it is in a DISPLAY, STRING, STOP, or UNSTRING statement), the length of the string is one occurrence of the ALL literal or one character in all other cases.

A figurative constant is valid wherever the word literal (or its abbreviation, "lit") appears in a general format or its associated rules. However, ZERO (ZEROS or ZEROES, plural) is the only valid figurative constant for literals restricted to numeric characters.

The actual characters associated with HIGH-VALUE, HIGH-VALUES, LOW-VALUE, and LOW-VALUES depend on the program collating sequence. For more information, see OBJECT-COMPUTER and SPECIAL-NAMES in Chapter 4: Environment Division.

### 1.2.4. PICTURE Character-Strings
A PICTURE character-string defines the size and category of an elementary data item. It can consist of the currency symbol ($) and certain combinations of characters in the COBOL character set. (See PICTURE.)

A punctuation character that is part of a PICTURE character-string is not considered to be a punctuation character. Instead, the compiler treats it as a symbol within the PICTURE character-string.

### 1.2.5. Separators

A **separator** delimits character-strings. It can be one character or two contiguous characters formed according to the rules in Table 1.9, “Separators”.

#### Table 1.9. Separators

<table>
<thead>
<tr>
<th>Separator</th>
<th>Usage Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>The space can be a separator or part of a separator.</td>
</tr>
<tr>
<td></td>
<td>• Where a space is used as a separator or part of a separator, more than one space can be used.</td>
</tr>
<tr>
<td></td>
<td>• A space can immediately precede any separator except:</td>
</tr>
<tr>
<td></td>
<td>• As specified by the rules for reference formats (see Section 1.3, “Source Reference Format”)</td>
</tr>
<tr>
<td></td>
<td>• The closing quotation mark of a nonnumeric literal; the space is then considered part of the nonnumeric literal rather than a separator.</td>
</tr>
<tr>
<td></td>
<td>• A space can immediately follow any separator except the opening quotation mark of a nonnumeric literal. After an opening quotation mark, the space is considered part of the nonnumeric literal rather than a separator.</td>
</tr>
<tr>
<td>Comma and Semicolon</td>
<td>The comma and semicolon are separators when they immediately precede a space. In this case, the comma and semicolon are interchangeable with each other and with the separator space. They can be used anywhere in a source program that a separator space can be used.</td>
</tr>
<tr>
<td>Period</td>
<td>The period is a separator when it immediately precedes a space or a return character. It can be used only where allowed by:</td>
</tr>
<tr>
<td></td>
<td>• Statement and sentence structure definitions (see Section 6.1, “Verbs, Statements, and Sentences”)</td>
</tr>
<tr>
<td></td>
<td>• Reference format rules (see Section 1.3, “Source Reference Format”)</td>
</tr>
<tr>
<td>Parentheses</td>
<td>Parentheses can be used only in balanced pairs of left and right parentheses to delimit:</td>
</tr>
<tr>
<td></td>
<td>• Subscripts</td>
</tr>
</tbody>
</table>
## 1.3. Source Reference Format

The VSI COBOL compiler recognizes two source program formats: ANSI and terminal.

- ANSI format conforms to the American National Standard COBOL reference format.

- Terminal format is a concise VSI specified format. It shortens source program lines by allowing horizontal tab characters and carriage returns. In terminal format, you do not use the ANSI format sequence numbers or identification area.
By default, the compiler expects terminal-format source lines. The compiler expects ANSI format only when the command line includes the `ansi` compiler option.

The reference format rules for spacing take precedence over all other spacing rules.

### 1.3.1. ANSI Format

The *ANSI source reference format* describes COBOL programs in terms of character positions on an input line. A source program line has 80 character positions as shown in Figure 1.1, “Source Program Line”.

**Figure 1.1. Source Program Line**

![Source Program Line](image)

**Margin L**

Immediately to the left of the leftmost character position.

**Margin C**

Between character positions 6 and 7.

**Margin A**

Between character positions 7 and 8.

**Margin B**

Between character positions 11 and 12.

**Margin R**

Between character positions 72 and 73.

**Sequence Number Area**

The six character positions between Margin L and Margin C. The contents can be any characters from the computer character set.

The compiler does not check the uniqueness of the contents. However, the compiler does check for the ascending sequence of the contents if the compiler command line includes the `sequence` compiler option.

**Indicator Area**

The seventh character position. The character in this position directs the compiler to interpret the source line in one of the following ways:

<table>
<thead>
<tr>
<th>Character</th>
<th>Source Line Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>space ( )</td>
<td>Default. The compiler processes the line as normal COBOL text.</td>
</tr>
<tr>
<td>hyphen (-)</td>
<td>Continuation line. The compiler processes the line as a continuation of the previous source line.</td>
</tr>
</tbody>
</table>
Overview of the COBOL Language

<table>
<thead>
<tr>
<th>Character</th>
<th>Source Line Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>asterisk (*)</td>
<td>Comment line. The compiler ignores the contents of the line. However, the source line appears on the program listing.</td>
</tr>
<tr>
<td>slash (/)</td>
<td>New listing page. The compiler treats the line as a comment line. However, it advances the program listing to the top of the next page before printing the line.</td>
</tr>
<tr>
<td>A-Z, a-z</td>
<td>Conditional compilation lines. The compiler processes the line as normal COBOL text if you specify the DEBUGGING MODE clause in the SOURCE-COMPUTER paragraph, or if you specify the conditionals compiler option in the command line. If you do not specify either, the compiler processes this line as a comment line.</td>
</tr>
</tbody>
</table>

**Area A**

The four character positions between Margin A and Margin B. Area A contains division headers, section headers, paragraph headers, paragraph-names, level indicators, and certain level-numbers.

**Area B**

The 61 character positions between Margin B and Margin R. Area B contains all other COBOL text.

**Identification Area**

The eight character positions immediately following Margin R. The compiler ignores the contents of the identification area. However, the contents appear on the source program listing.

**Line Continuation**

Sentences, entries, phrases, and clauses that continue in Area B of subsequent lines are called continuation lines. The line being continued is called the continued line.

A hyphen in a line's indicator area causes its first nonblank character in Area B to be the immediate successor of the last nonblank character of the preceding line. This continuation excludes intervening comment lines and blank lines.

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

If the indicator area is blank:

- The compiler treats the first nonblank character on the line as if it followed a space.
- The compiler treats the last nonblank character on the preceding line as if it preceded a space.

**ANSI Format Example**

001010 01 NUMERIC-CONTINUATION.
001020 03 NUMERIC-LITERAL PIC 9(16) VALUE IS 123
Lines 001020 and 001030 show continuation of a numeric literal. Lines 001050 and 001060 continue a nonnumeric literal. A sentence that spans four lines begins on line 001090.

Blank Lines

A blank line contains no characters other than spaces between Margin C and Margin R. Blank lines can be anywhere in a source program or library text.

Comment Lines

A comment line is any source line with an asterisk (*) or slash (/) in its indicator area. Area A and Area B can contain any characters from the computer character set. Comment lines can be anywhere in a source program or library text.

Conditional Compilation Lines

A conditional compilation line is any source line after the OBJECT COMPUTER paragraph that includes one of these uppercase or lowercase alphabetic characters in its indicator area: A to Z, a to z. The compiler processes the line as normal COBOL text if you specify the DEBUGGING MODE clause in the SOURCE COMPUTER paragraph.

The compiler processes the line as normal COBOL text if you include the appropriate conditionals compiler option in the command line.

If you specify neither, the compiler processes this line as a comment line.

Lines conditioned by one letter can be compiled or treated as comments independently of other conditional compilation lines. On OpenVMS systems, for instance, if you compile with /CONDITIONALS=(A,B), lines conditioned with A and B compile while those conditioned by other letters are treated as comments.

See Chapter 8, Source Text Manipulation for additional information on the interaction between conditional compilation lines and the COPY statement.

Pseudo-Text

Pseudo-text character-strings and separators can start in either Area A or Area B. However, if there is a hyphen in the indicator area of a line that follows the opening pseudo-text delimiter, Area A of the line must be blank.

The normal rules for line continuation apply to the formation of text-words.

Pseudo-text is described in Chapter 8, Source Text Manipulation.

Short Lines and Tab Characters

If the source program input medium is not punched cards, carriage return and horizontal tab characters can shorten source lines.
The compiler recognizes the end of the input line as Margin R. Tab characters, other than those in nonnumeric
literals, cause the compiler to generate enough space characters to position the next character at the next tab stop.
The compiler's tab stops are at character positions 8, 12, 20, 28, 36, 44, 52, 60, 68, and 76.

The following example shows how the compiler interprets carriage return and horizontal tab characters in a source
program:

**Shortened ANSI Format Source Line**

```
000100*The following record description shows the source line format
       Return
000110 01 Tab RECORD-A. Return
000120 Tab Tab03 GROUP-A. Return
000130 Tab Tab Tab05 ITEM-A Tab PIC X(10). Return
000140* Tab The tab character in the nonnumeric literal Return
000150* Tab on the next line is stored as one character Return
000160 Tab Tab Tab05
          ITEM-B Tab PIC X VALUE IS " Tab". Return
000170 Tab Tab03 ITEM-C Tab Tab PIC X(10). Return
000180D01 Tab RECB REDEFINES RECORD-A Tab PIC X(21). Return
```

**Source Line as Interpreted by the Compiler**

```
000100*The following record description shows the source line format
000110 01 RECORD-A.
000120 03 GROUP-A.
000130 05 ITEM-A PIC X(10).
000140* The tab character in the nonnumeric literal
000150* on the next line is stored as one character
000160 05 ITEM-B PIC X VALUE IS " Tab".
000170 03 ITEM-C PIC X(10).
000180D01 RECB REDEFINES RECORD-A PIC X(21).
```

Use more tab characters only when necessary. Compiler error diagnostics result if you use tab characters beyond the
permissible character positions for a COBOL statement or entry. The following example shows how the compiler
treats source program lines 000004 and 000005. Line 000004: contains one too many tab characters, which places
paragraph-name P0 out of Area A.

**Shortened ANSI Format Source Line**

```
000001 TabIDENTIFICATION DIVISION.
000002 TabPROGRAM-ID. ANSI-TEST.
000003 TabPROCEDURE DIVISION.
000004 TabTabP0.
000005 TabTabSTOP RUN.
```

**Listing File Result on OpenVMS Alpha, l64**

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID. ANSI-TEST.
000003 PROCEDURE DIVISION.
000004 P0.
............^%COBOL-F-UNDEFSYM, Undefined name
    at line number 4 in file DISK:[DIRECTORY]ANSI.COB;1
000005 STOP RUN.
............^%COBOL-W-SYN6, Missing paragraph header
Overview of the COBOL Language

Listing File Result on Tru64 UNIX

000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID. ANSI-TEST.
000003 PROCEDURE DIVISION.
cobol: Severe: dwork/t.cob, line 4: Undefined name
000004 P0.
--------^
cobol: Warning: dwork/t.cob, line 5: Missing paragraph header
000005 STOP RUN.
--------^  

Note

The previous error messages have no additional online explanations. If a diagnostic message has a further explanation, an asterisk (*) is displayed (to the left of the error message). On OpenVMS Alpha and I64 systems, the VSI COBOL online help file lists and describes error messages that have further explanations.

1.3.2. Terminal Format

The VSI COBOL terminal format shortens program preparation time and reduces storage space for source programs. This format eliminates the sequence number and identification areas. It also combines the indicator area with Area A. Except for the differences described in this section, the rules for ANSI format also apply to terminal-format source programs.

In terminal format, the compiler recognizes the following valid indicator area characters in the first character position:

(-) hyphen
(*) asterisk
(/) slash

The compiler also recognizes the following conditional compilation line characters as valid indicator area characters in the first and second character positions:

(\x) backslash and x

where x can be any uppercase or lowercase alphabetic character.

Area A then begins in character position 2 (or 3 if using \x). Otherwise, Area A begins in the first character position.

Area B begins four character positions to the right of the beginning of Area A. It ends when the compiler detects a carriage return, or at Margin R.

The maximum length of a terminal-format source line is 256 characters. The compiler's tab stops are immediately to the right of Margin B, and every eight character positions to the right, until the end of the line.

Note

The maximum length of the source line on the program listing is 125 characters, including the sequence field. The compiler processes the complete source line but displays only the first 125 characters on the listing. It also replaces all nonprintable ASCII characters with periods (or other symbols depending on the device) in the listing file. (Refer to the VSI COBOL User Manual.)

The following example shows source lines in terminal format. It is equivalent to the ANSI-format source line examples in the previous section.
*The following record description shows the source line format Return 01 Tab RECORD-A. Return Tab03 GROUP-A. Return Tab Tab05 ITEM-A Tab PIC X(10). Return * Tab The tab character in the nonnumeric literal Return * Tab on the next line is stored as one character Return Tab Tab05 ITEM-B Tab PIC X VALUE IS " Tab". Return Tab03 ITEM-C Tab Tab PIC X(10). Return \D01 Tab RECB REDEFINES RECORD-A Tab PIC X(21). Return

1.4. Sample Entry Format

The following format is used to describe most entries in this manual. Each COBOL division or major topic begins a new chapter and each entry begins on a new page. The entries are in functional or alphabetical order.

Entry-Name

Function

The function paragraph describes the function or the effect of the entry.

General Format

A general format shows the specific arrangement of elements in the entry. If there is more than one arrangement, the formats are numbered. All clauses (mandatory and optional) must be used in the sequence shown in the format. However, the syntax rules sometimes allow exceptions.

generic-term

Following the general format are definitions of its generic terms. These terms appear in the rules in italic type.

Syntax Rules

Syntax rules define or clarify the arrangement of words or elements. They can also impose further restrictions or relax restrictions implied by the general format.

General Rules

General rules define or clarify the meaning (or relationship of meanings) of an element or set of elements. They also define the semantics of an entry, describing its effects on program compilation or execution.

Technical Notes

Technical notes describe, in system-specific terms, any system-specific behavior, and any other VSI COBOL behavior of note not described in the rules. They define relationships between the COBOL program and the operating system and its components.

Additional References

Additional references point to other relevant information in this manual, the VSI COBOL User Manual, and other VSI documentation sets.

Examples

Examples show the use of a statement, clause, or other entry. The VSI COBOL User Manual contains other examples in application contexts.

The following example shows a general format:
Overview of the COBOL Language

General Format

Additional References

• Chapter 3: Identification Division
• Chapter 4: Environment Division
• Chapter 5: Data Division
• Chapter 6: Procedure Division
Chapter 2. Organization of a COBOL Program

A COBOL source program is a syntactically correct set of COBOL statements that:

• Mark the beginning of the program
• Describe its physical environment
• Describe the data the program creates, receives as input, manipulates, and produces as output
• Specify the processing of the program's files and data

General Format

[identification-division]
represents a COBOL Identification Division.

[environment-division]
represents a COBOL Environment Division.

[data-division]
represents a COBOL Data Division.

[procedure-division]
represents a COBOL Procedure Division.

[source-program]
represents a contained (nested) COBOL source program. A COBOL source program may be nested; more than one source program may be present in a single source file.

[end-program-header]
represents a COBOL END PROGRAM header.

**Syntax Rule**

The end-program-header must be present if either:

1. The COBOL source program contains one or more other COBOL source programs.
2. The COBOL source program is contained within another COBOL source program.
3. The COBOL source program precedes another separately compiled program.

**General Rules**

1. The appropriate division header indicates the beginning of a division.
2. The following indicates the end of a division:
   a. Another division header
   b. An Identification Division header that indicates the start of another source program
   c. The end-program-header
   d. The physical position at which no further source lines occur
3. A COBOL source program may contain other COBOL source programs.
4. A COBOL source program that is directly or indirectly contained within another program is called a contained or nested program. It may reference certain resources in the containing program.
5. A separately compiled program has a nesting level number of 1. If this program contains other source-programs, it is the outermost containing program.
6. A contained program has a nesting level number greater than 1.

**Additional References**

- Identification Division
- Environment Division
- Data Division
- Procedure Division
- END PROGRAM Header

**Program Structure**

Figure 2.1, “Structure of a COBOL Program” shows the basic structure of a COBOL program, which is organized in divisions, sections, paragraphs, sentences, and entries.
**Figure 2.1. Structure of a COBOL Program**

IDENTIFICATION DIVISION.
PROGRAM-ID. program-name.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
OPTIONS.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.
DATA DIVISION.
SUBSCHEMA SECTION.
  subschema entries and keeplist entries
FILE SECTION.
  file and record description entries
  report file description entries
  sort-merge file and record description entries
WORKING- STORAGE SECTION.
  record description entries
LINKAGE SECTION.
  record description entries
REPORT SECTION.
  report and report group description entries.
SCREEN SECTION. (Alpha, I64)
  screen description entries (Alpha, I64)
PROCEDURE DIVISION.
DECLARATIVES.
  sections
    paragraphs
    sentences
END DECLARATIVES.
.
.
sections
  paragraphs
  sentences
.
.
END PROGRAM header

**Division Header**

A division header identifies and marks the beginning of a division. It is a specific combination of reserved words followed by a separator period. Division headers start in Area A.

Except for the COPY and REPLACE statements, and the END PROGRAM header (see END PROGRAM in Chapter 6, Procedure Division), the statements, entries, paragraphs, and sections of a COBOL source program are grouped into four divisions in this order:
1. **IDENTIFICATION DIVISION.**

2. **ENVIRONMENT DIVISION.**

3. **DATA DIVISION.**

4. **PROCEDURE DIVISION.**

The end of a COBOL source program is indicated either by the END PROGRAM header (END PROGRAM) or by the end of that program's Procedure Division.

Only these items can immediately follow a division header:

- Another division header
- A section header
- A paragraph header or paragraph-name
- A comment line
- A blank line
- A DECLARATIVES header for the USE procedure sections (after the PROCEDURE DIVISION header only)
- A PROGRAM-ID paragraph (after the IDENTIFICATION DIVISION header only)

Only this item can immediately follow a DECLARATIVES header:

- A section header for a USE procedure

**Note**

The PROCEDURE DIVISION header can contain a USING and GIVING phrase. (See Section 6.8, “General Formats and Rules for Statements”.)

**Section Header**

A *section header* identifies and marks the beginning of a section in the Environment, Data, and Procedure Divisions. In the Environment and Data Divisions, a section header is a specific combination of reserved words followed by a separator period. In the Procedure Division, a section header is a user-defined word followed by the word SECTION (and an optional segment-number). A separator period always follows a section header. Section headers start in Area A.

The valid section headers follow for each division.

In the Environment Division:

**CONFIGURATION SECTION.**
**INPUT-OUTPUT SECTION.**

In the Data Division:

**FILE SECTION.**
**WORKING-STORAGE SECTION.**
**LINKAGE SECTION.**
**REPORT SECTION.**
**SCREEN SECTION.** (Alpha, I64)

In the Procedure Division:
user-name SECTION [ segment-number ].

Only these items can immediately follow a section header:

• A division header
• Another section header
• A paragraph header or paragraph-name
• A comment line
• A USE statement (in the DECLARATIVES part of the Procedure Division only)
• A blank line
• A DATA DIVISION entry (in the Data Division)

**Paragraph, Paragraph Header, Paragraph-Name**

A paragraph consists of a paragraph header or paragraph-name (depending on the division) followed by zero, one, or more entries (or sentences).

A paragraph header is a reserved word followed by a separator period. Paragraph headers identify paragraphs in the Identification and Environment Divisions.

The paragraph headers are as follows:

<table>
<thead>
<tr>
<th>Identification Division</th>
<th>Environment Division</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM-ID.</td>
<td>SOURCE-COMPUTER.</td>
</tr>
<tr>
<td>AUTHOR.</td>
<td>OBJECT-COMPUTER.</td>
</tr>
<tr>
<td>INSTALLATION.</td>
<td>SPECIAL-NAMES.</td>
</tr>
<tr>
<td>DATE-WRITTEN.</td>
<td>FILE-CONTROL.</td>
</tr>
<tr>
<td>DATE-COMPILED.</td>
<td>I-O-CONTROL.</td>
</tr>
<tr>
<td>SECURITY.</td>
<td></td>
</tr>
<tr>
<td>OPTIONS.</td>
<td></td>
</tr>
</tbody>
</table>

A paragraph-name is a user-defined word followed by a separator period. Paragraph-names identify paragraphs in the Procedure Division.

Paragraph headers and paragraph-names start in Area A of any line after the first line of a division or section.

The first entry or sentence of a paragraph begins either:

• On the same line as the paragraph header or paragraph-name
• In Area B of the next nonblank line that is not a comment line

Successive sentences or entries begin in Area B of either:

• The same line as the preceding entry or sentence
• The next nonblank line that is not a comment line

**Data Division Entries**

A Data Division entry begins with a level indicator or level-number and is followed, in order, by:
1. A space
2. The name of a data item, file connector, or screen item
3. A sequence of independent descriptive clauses
4. A separator period

The level indicators are as follows:

- FD (for file description entries)
- SD (for sort-merge file description entries)
- RD (for report file description entries)

Level indicators can begin anywhere to the right of Area A.

Entries that begin with level-numbers are called either data description or screen description entries, depending on their context. The level-number values are 01 to 49, 66, 77, and 88 for data description items and 01 to 49 for screen description entries. Level-numbers 01 to 09 can be represented as one- or two-digit numbers.

All data description entries and screen description entries can begin anywhere to the right of Margin A. However, indentation has no effect on level-number magnitude; it merely enhances readability.

**Declaratives**

*Declaratives* specify USE procedures to be executed only when certain conditions occur. You must write USE procedures at the beginning of the Procedure Division in consecutive sections. The key word DECLARATIVES begins the DECLARATIVES part of the Procedure Division; the pair of key words END DECLARATIVES ends it. Each of these reserved word phrases must be on a line by itself, starting in Area A; and be followed by a separator period. For example:

```
PROCEDURE DIVISION.
DECLARATIVES.
IOERROR SECTION.
USE AFTER ...
PAR-1.
.
.
.
END DECLARATIVES.
```

When you specify USE procedures, you must divide the remainder of the Procedure Division into sections.
Chapter 3. Identification Division

Function

The Identification Division marks the beginning of a COBOL program. It also identifies a program and its source listing.

General Format

* These paragraphs are not described in individual entries; they follow the same format as the AUTHOR paragraph and are for documentation only.

Syntax Rules

1. The Identification Division must be the first entry in a COBOL program.

2. The Identification Division must begin with the IDENTIFICATION DIVISION header. The header consists of the reserved words IDENTIFICATION DIVISION followed by a separator period.

3. The PROGRAM-ID paragraph must immediately follow the IDENTIFICATION DIVISION header.

PROGRAM-ID

PROGRAM-ID — The PROGRAM-ID paragraph identifies a program and assigns selected program attributes.
General Format

[program-name]

is a user-defined word that names the program.

Syntax Rules

1. The PROGRAM-ID paragraph must be present in every program.

2. program-name must contain 1 to 31 characters and follow the rules for user-defined words.

3. Programs contained within a separately compiled program must have a unique program-name.

4. The optional COMMON clause may be used only if the program is contained within another program.

5. ident-string must be a nonnumeric literal 1 to 31 characters in length.

6. The optional IDENT clause cannot be used in a contained program.

General Rules

1. program-name is a user-defined word that identifies a COBOL program and its source listing. It appears as the first word in the first line of every page in the compiler source listing.

2. program-name represents the object program entry point.

3. If an executable image includes more than one separately compiled program, each separately compiled program must have a unique program-name.
4. The COMMON clause specifies a common program. A common program is contained within another program but may be called from programs other than that directly containing it.

5. Files associated with a called program's internal file connectors are not in the open mode:
   a. The first time the program is called
   b. The first time the program is called after execution of a CANCEL statement referring to the program
   c. Every time the program is called, if it has the INITIAL attribute

   On all other entries, the status and positioning of files in a called program are the same as when the program last exited.

6. The INITIAL clause specifies an initial program. Whenever the program is called, it and any programs contained within it are placed in their initial state, and the internal data in each program is initialized.

7. On OpenVMS, the IDENT clause specifies a literal string that is used for identification purposes. This string is written to the object file as the "module version."

   When the /ANALYSIS_DATA qualifier is included on the COBOL command, the string is written to the analysis data file as the module ident.

8. On Tru64 UNIX systems, program-name is case-sensitive. By default, program-name is converted to lowercase for all separately compiled program units. Any calls from other programs (VSI COBOL as well as other languages) must specify the routine to be called in lowercase.

   However, if the names option is set to uppercase on the command line, calls from other programs must specify the routine to be called in uppercase. If the names option is set to as_is, the effect on program-name is as if uppercase were specified. (The as_is setting is used for calling non-COBOL programs with mixed case.)

Additional Reference

See Section 6.2.6: Scope of Names.

Examples

PROGRAM-ID. PROGA.
PROGRAM-ID. SUBR1 INITIAL.
PROGRAM-ID. COMPUTE-PAY WITH IDENT "JOB6a-V1.1". (OpenVMS)
PROGRAM-ID.
WRITEMASTERREPORT.
PROGRAM-ID. PAYROLL IS COMMON.
Identification

AUTHOR

AUTHOR — The AUTHOR paragraph is for documentation only.
General Format

[comment-entry] is a user-supplied comment about the program’s author.

Syntax Rules

1. *comment-entry* can consist of any combination of characters from the computer character set.

2. *comment-entry* can span several lines in Area B. However, they cannot be continued by using a hyphen in the indicator area.

3. The end of *comment-entry* is the line before the next entry in Area A.

Examples

AUTHOR. JOHN SMITH.

AUTHOR. This program was written by John Smith
1226 Main St.
Merrimack, NH 03054

AUTHOR.

DATE-COMPILED

DATE-COMPILED — The DATE-COMPILED paragraph provides the compilation date in the source program listing file.
[comment-entry]

is user-supplied information about the date compiled.

**Syntax Rules**

1. `comment-entry` can consist of any combination of characters from the computer character set.

2. `comment-entry` can span several lines in Area B. However, it cannot be continued by using a hyphen in the indicator area.

3. The end of `comment-entry` is the line before the next entry in Area A.

**General Rule**

The paragraph-name DATE-COMPILED causes the current date to be inserted in your source program listing during compilation. Therefore, if a DATE-COMPILED paragraph is present in your source program, it will be replaced with a paragraph of the following form:

`DATE-COMPILED. dd-mmm-yyyy.`

**OPTIONS**

OPTIONS — The OPTIONS paragraph specifies information for use by the compiler in generating executable code for a source unit.
General Format

[arithmetic-clause]

specifies the method used in developing the intermediate results. The format is:
Syntax Rule

The period appearing in the general format after the arithmetic-clause may be omitted if the arithmetic-clause is not specified.

General Rules

1. The ARITHMETIC clause in the OPTIONS paragraph applies to the source element in which it is specified and to all source elements contained in that source element unless overridden by an ARITHMETIC clause in an OPTIONS paragraph in a contained source element.

2. If the NATIVE phrase is specified, the techniques used in handling arithmetic expressions and arithmetic statements shall be those specified for native arithmetic in the appendix on compatibility in the VSI COBOL User Manual.

3. If the STANDARD phrase is specified, the techniques used in handling arithmetic expressions and arithmetic statements shall be those specified for standard arithmetic in the ANSI Standard for COBOL. (Refer to the appendix on compatibility in the VSI COBOL User Manual.)

4. If the ARITHMETIC clause is not specified in this source element or a containing source element, it is as if the ARITHMETIC clause were specified with the NATIVE phrase.
Chapter 4. Environment Division

Function

The Environment Division describes the program's physical environment. It also specifies input-output control and describes special control techniques and hardware characteristics.

The Environment Division can contain two sections:

• Configuration Section (see Section : 4.1 CONFIGURATION Section)
• Input-Output Section (see : INPUT-OUTPUT)

Syntax Rules

1. The Environment Division follows the Identification Division.
2. The general format defines the order of appearance of Environment Division entries.
3. A contained program cannot include a Configuration Section.

General Rule

Explicit or implicit Configuration Section entries in a program containing other programs apply to each contained program.

4.1 CONFIGURATION Section
The Configuration Section can contain three paragraphs:

- SOURCE-COMPUTER paragraph (see SOURCE-COMPUTER)
- OBJECT-COMPUTER paragraph (see OBJECT-COMPUTER)
- SPECIAL-NAMES paragraph (see SPECIAL-NAMES)

The Configuration Section must not be stated in a program that is contained within another program. If Configuration Section entries are stated in a program that contains other programs, they apply to each contained program.

### SOURCE-COMPUTER

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

**Description**

[computer-type]

is a user-defined word that names the computer.

**Syntax Rule**

*ALPHA* and *I64* are system-names. They are not reserved words, and are for documentation only.

**General Rules**

1. If the WITH DEBUGGING MODE clause is not used, this paragraph is for documentation only.
2. All clauses of the SOURCE-COMPUTER paragraph apply to the program that specifies them. They also apply to any program contained within that program.
3. If you include the WITH DEBUGGING MODE clause in a program, or if you specify the *conditionals* command-line option, all conditional compilation lines are compiled. Otherwise, the compiler treats all conditional compilation lines as comment lines. (See Section 1.3.1, “ANSI Format” for additional information about source line interpretation.)

### OBJECT-COMPUTER

OBJECT-COMPUTER — The OBJECT-COMPUTER paragraph describes the computer on which the program is to execute.
Description

[computer-type]

is a user-defined word that names the computer.

[integer]

is a numeric literal that has no digits to the right of the assumed decimal point.

[alpha-name]

is the name of a collating sequence defined in the ALPHABET clause of the SPECIAL-NAMES paragraph.

[segment-number]

is an integer from 1 to 49.

Syntax Rule

*ALPHA* and *I64* are system-names. They are not reserved words, and are for documentation only.

General Rules

1. All clauses of the OBJECT-COMPUTER paragraph apply to the program that explicitly or implicitly specifies them. They also apply to any program contained within that program.

2.

3.
The PROGRAM COLLATING SEQUENCE clause causes the program to use the collating sequence of *alpha-name* to determine the truth value of nonnumeric comparisons in:

- Relation conditions
- Condition-name conditions
- Report description entries, the CONTROL clause

4. The PROGRAM COLLATING SEQUENCE clause also applies to nonnumeric merge and sort keys. However, the COLLATING SEQUENCE phrase in a MERGE or SORT statement takes precedence over the PROGRAM COLLATING SEQUENCE clause.

5. If there is no PROGRAM COLLATING SEQUENCE clause, the program uses the NATIVE collating sequence.

6. The SEGMENT-LIMIT clause is for documentation only.

**Additional References**

- SPECIAL-NAMES Paragraph
- SPECIAL-NAMES Paragraph
- Section 6.5.1: Relation Conditions
- Section 6.5.3: Condition-Name Condition
- Section 6.7: Segmentation

Additionally, refer to the information on SORT and MERGE statements in the VSI COBOL User Manual.

**Examples**

1. Computer name only:

   ```
   OBJECT-COMPUTER. Alpha.
   ```

2. No computer name (if the computer is not specified, then no other clause can appear):

   ```
   OBJECT-COMPUTER.
   ```

3. With PROGRAM COLLATING SEQUENCE clause:

   ```
   OBJECT-COMPUTER. Alpha
   PROGRAM COLLATING SEQUENCE IS ALPH-A.
   ```

   The SPECIAL-NAMES paragraph must define ALPH-A.

4. With PROGRAM COLLATING SEQUENCE clause:

   ```
   OBJECT-COMPUTER. Alpha
   SEQUENCE IS EBCDIC.
   ```

   The SPECIAL-NAMES paragraph must define EBCDIC.

   If EBCDIC refers to the EBCDIC collating sequence, the SPECIAL-NAMES paragraph must contain the following clause:

   ```
   ALPHABET EBCDIC IS EBCDIC
   ```
SPECIAL-NAMES

SPECIAL-NAMES — The SPECIAL-NAMES paragraph: (1) associates compiler features and logical names (on OpenVMS systems) or environment variables (on Tru64 UNIX systems) with user-defined mnemonic-names, (2) provides a way to reference command-line arguments and (on Tru64 UNIX) environment variables or (on OpenVMS) logical names with user-defined mnemonic names, (3) defines symbolic-characters, (4) specifies the currency sign, (5) selects the decimal point, (6) relates alphabet-names to character sets or collating sequences, (7) relates class-names to character sets, (8) provides for cursor positioning for an ACCEPT (Format 5) statement, and (9) provides information on the cause of termination of an ACCEPT (Format 5) statement.

Synopsis
Description

[device-name]

is a user-defined word for a device. Only the ACCEPT and DISPLAY statements can refer to it.

[argument-number]

is a user-defined word that contains the current argument position indicator number when used with DISPLAY, or the count of command line arguments when used with ACCEPT. Only the ACCEPT and DISPLAY statements can refer to it.

[argument-value]

is a user-defined word that contains the value of the current command line argument as indicated by the current ARGUMENT-NUMBER. Only the ACCEPT and DISPLAY statements can refer to it.

[environment-name]

is a user-defined word that contains the name of an environment variable or system logical. Only the ACCEPT and DISPLAY statements can refer to it.

[environment-value]

is a user-defined word that contains the value of the environment variable or logical named by the current ENVIRONMENT-NAME. Only the ACCEPT and DISPLAY statements can refer to it.

[top-of-page-name]

is a user-defined word for the top of a page. Only the WRITE statement can refer to it.

[switch-num]
is the number of a program switch. Its value can range from 1 to 16.

[switch-name]

is a mnemonic-name for the program switch.

[cond-name]

is a condition-name for the on or off status of the switch. It always possesses the global attribute. Its truth value is true when the STATUS phrase matches the status of the switch, false when it does not.

[alpha-name]

is the user-defined word for a character set, collating sequence, or both. It always possesses the global attribute.

[first-literal]

is a literal. It specifies either: (1) the value of one or more alphabet characters, or (2) the first in a range of values.

[last-literal]

is a literal. It specifies the last in a range of values.

[lit]

is a literal. It specifies an alphabet character value.

[symbol-char]

is a user-defined word that names the symbolic-character. It always possesses the global attribute. The same symbol-char cannot appear more than once in the SYMBOLIC CHARACTERS clause.

[char-val]

is an integer that indicates the ordinal position of a character in the native character set.

[class-name]

is the user-defined word for a class. It always possesses a global attribute.

[char]

is a one-character nonnumeric literal that specifies the currency symbol. It cannot be a symbolic-character or figurative constant.

[literal-7 (Alpha, I64)]

is an alphanumeric literal. It cannot be a figurative constant.

[literal-8 (Alpha, I64)]

is an alphanumeric literal consisting of a single character. It cannot be a figurative constant. No two occurrences of literal-8 can have the same value.

[cursor-position (Alpha, I64)]

is a data item declared in the Working-Storage Section of the program. It is either an elementary unsigned numeric integer either four or six characters in length, described as USAGE IS DISPLAY, or a group item either four or six characters in length, consisting of two elementary unsigned data items.

[crt-status-code (Alpha, I64)]

is a group data item three characters in length, declared in the Working-Storage Section of the program.
## Syntax Rules

1. In the first-literal phrase of the ALPHABET or CLASS clauses:
   - If *alpha-name* is in the PROGRAM COLLATING SEQUENCE clause, the ALPHABET clause cannot specify any character more than once.
   - If the ALSO or THRU phrase appears, *first-literal* must be one character long.
   - Numeric literals must be unsigned integers from 1 to 256.
   - If *last-literal* or *lit* is nonnumeric, it must be one character long.
   - THRU and THROUGH are equivalent.

2. If the *first-literal* phrase appears, *alpha-name* cannot be referenced in a CODE-SET clause.

3. The following are accessible only by ACCEPT and DISPLAY statements:
   - argument-count
   - argument-value
   - environment-name
   - environment-value

## General Rules

1. All clauses of the SPECIAL-NAMES paragraph apply to the program defining them and to all programs contained within that program.

### device-name Clause

1. The device-name clause associates a device with a user-defined word (*device-name*).

   On Tru64 UNIX, the device name is derived from an environment variable, if that environment variable exists. Otherwise, the defaults are as follows:

<table>
<thead>
<tr>
<th>System-Name</th>
<th>Tru64 UNIX Environment Variable</th>
<th>Tru64 UNIX Default File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARD-READER</td>
<td>COBOL_CARDREADER</td>
<td>stdin</td>
</tr>
<tr>
<td>PAPER-TAPE-READER</td>
<td>COBOL_PAPERTAPEREADER</td>
<td>stdin</td>
</tr>
<tr>
<td>CONSOLE</td>
<td>COBOL_CONSOLE</td>
<td>stderr</td>
</tr>
<tr>
<td>LINE-PRINTER</td>
<td>COBOL_LINEPRINTER</td>
<td>stdout</td>
</tr>
<tr>
<td>PAPER-TAPE-PUNCH</td>
<td>COBOL_PAPERTAPEPUNCH</td>
<td>stdout</td>
</tr>
<tr>
<td>SYSIN</td>
<td>COBOL_INPUT</td>
<td>stdin</td>
</tr>
<tr>
<td>SYSOUT</td>
<td>COBOL_OUTPUT</td>
<td>stdout</td>
</tr>
<tr>
<td>SYSERR</td>
<td>COBOL_ERROR</td>
<td>stderr</td>
</tr>
</tbody>
</table>

   The input device for the ACCEPT statement is derived from COBOL_INPUT, if defined, and defaults to stdin. The output device for the DISPLAY statement is derived from COBOL_OUTPUT, if defined, and defaults to stdout.

   On OpenVMS, the file-name is derived from a logical name if that logical name exists. Otherwise, the defaults are as follows:

<table>
<thead>
<tr>
<th>System-Name</th>
<th>OpenVMS Logical Name</th>
<th>OpenVMS Default File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARD-READER</td>
<td>COBS_CARDREADER</td>
<td>SYSSINPUT</td>
</tr>
</tbody>
</table>
The input device for the ACCEPT statement is derived from COBSINPUT, if defined, and defaults to SYS$INPUT. The output device for the DISPLAY statement is derived from COBSOUTPUT, if defined, and defaults to SYS$OUTPUT. (See the ACCEPT and DISPLAY statements in Chapter 6: Procedure Division, and refer to the VSI COBOL User Manual for more information.)

**top-of-page-name Clause**

1. The system-name C01 refers to the first line of a logical page. Only the ADVANCING phrase of the WRITE statement can refer to the top-of-page-name equated to C01. (See the WRITE statement in Chapter 6: Procedure Division.)

**SWITCH Clause**

1. The ON STATUS (or OFF STATUS) phrase of the SWITCH clause associates the status of switch-name with a corresponding cond-name. The program uses a switch-status condition in the Procedure Division to test the switch.

   Switches can also be read from the OpenVMS logical name COB$SWITCHES or the Tru64 UNIX environment variable COBOL SWITCHES.

   The compiler interprets SWITCH n and SWITCH-n (where n represents a number from 1 to 8) as identical clauses. For example, SWITCH 1 is equivalent to SWITCH-1.

   Refer to the VSI COBOL User Manual for more information on using switches.

**ALPHABET Clause**

1. The ALPHABET clause relates a name to a character code set, collating sequence, or both.

   The ALPHABET clause specifies:

   - A character code set, when alpha-name is in a CODE-SET clause in the FILE-CONTROL paragraph or file description entry.

   - A collating sequence, when alpha-name is in: (1) the PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph or (2) the COLLATING SEQUENCE phrase of a SORT or MERGE statement.

2.

3. STANDARD-1 refers to the ASCII character set.

4. STANDARD-2 refers to the international version of the ISO 7-bit code. It is defined in International Standard 646, “7-Bit Coded Character Set for Information Processing Interchange.”

5. NATIVE refers to the native character set. It consists of 256 characters. The lowest-valued 128 characters are the ASCII character set. The highest-valued 128 characters are reserved for later standardization and definition by VSI.
6. EBCDIC refers to the EBCDIC character set or collating sequence. It is defined in Appendix B: Character Sets.

7. The character with the highest ordinal position in the program collating sequence equals the figurative constant HIGH-VALUE, except when this figurative constant is specified as a literal in the SPECIAL-NAMES paragraph. If more than one character has the highest position, HIGH-VALUE is the last character you specify.

8. The character with the lowest ordinal position in the program collating sequence equals the figurative constant LOW-VALUE, except when this figurative constant is specified as a literal in the SPECIAL-NAMES paragraph. If more than one character has the lowest position, LOW-VALUE is the first character you specify.

**Literals in the ALPHABET Clause**

1. The value of each numeric literal specifies the ordinal number of a character in the native character set. For example, 66 refers to the ASCII character A.

2. The value of each nonnumeric literal specifies the actual character in the native character set.

3. If the literal contains more than one character, the compiler interprets each character from left to right. It assigns each a successive ascending position in the collating sequence or character code set.

4. The order of appearance of literals in the ALPHABET clause specifies each character's ordinal number in ascending sequence. If the ALPHABET clause defines a character code set, the ordinal number identifies the character's relative position in the set.

5. Any unspecified characters in the native collating sequence have higher positions in the new collating sequence than all specified characters. The relative order of the unspecified characters is the same as in the native collating sequence.

   For example, the following clauses are equivalent:

   ```plaintext
   ALPHABET XYZ IS 2 4
   ALPHABET XYZ IS 2 4 1 3 5 6 7
   ALPHABET XYZ IS 2 4
   ```

**THROUGH Phrase**

1. The THROUGH phrase specifies a set of contiguous characters in the native character set. The first character is `first-literal`; the last character is `last-literal`.

2. The compiler assigns each character in the set a successive ascending position in the collating sequence or character code set.

3. The THROUGH phrase can specify the set of contiguous characters in either ascending or descending order. For example, “L” THRU “H” assigns successively higher numbers to L, K, J, I, and H.

4. The ALSO phrase assigns `first-literal` and each `lit` to the same position in the collating sequence or character code set. For example, “A” ALSO “$” causes the characters A and $ to be equivalent in comparisons when the associated `alpha-name` is in the PROGRAM COLLATING SEQUENCE clause.

**SYMBOLIC CHARACTERS Clause**

1. Each `symbol-char` corresponds to the `char-val` in the same relative position. In the following example, CARRIAGE-RET corresponds to 14 and ESCAPE to 28:

   ```plaintext
   SYMBOLIC CHARACTERS CARRIAGE-RET ESCAPE ARE 14 28
   ```

2. If the `IN` phrase is not specified, `symbol-char` represents the character, in the native character set, that has the ordinal position specified by `char-val`.

48
Note

The ordinal position is one greater than the internal representation of the character. For example, the character A is in ordinal position 66. Its internal representation is decimal 65 (hexadecimal 41).

3. If the IN phrase is specified, char-val represents the character that has the ordinal position specified by the IN alpha-name phrase.

CLASS Clause

1. The CLASS clause relates a name to a specified set of characters in that clause. class-name can be referenced only in a class condition. The characters specified by the values of the literals in this clause define the set of characters of which this class-name consists.

2. The value of each numeric literal specifies the ordinal number of a character in the native character set. This value must not exceed the value that represents the number of characters in the native character set.

3. The value of each nonnumeric literal specifies the actual character in the native character set. If the nonnumeric literal contains multiple characters, each character in the literal is included in the set of characters identified by class-name.

4. The THROUGH phrase specifies a set of contiguous characters in the native character set. The first character is first-literal; the last character is last-literal. The characters specified by a given THROUGH phrase can be specified in ascending or descending order.

CURRENCY SIGN Clause

1. In the CURRENCY SIGN clause, char specifies the PICTURE clause currency symbol. It can be any printable character from the computer character set except:
   - 0 through 9
   - A, B, C, D, P, R, S, V, X, Z, the lowercase characters a to z, or the space
   - Asterisk (*), plus sign (+), minus sign (-), comma (,), period (.), semicolon (;), quotation mark ("), equal sign (=), slash (/), left parenthesis ( ( ), or right parenthesis ( )

2. The CURRENCY SIGN clause cannot contain a symbolic-character or figurative constant.

3. If there is no CURRENCY SIGN clause, the default currency sign used for the PICTURE clause is the "$" symbol.

   On OpenVMS, if you define the logical name SYSSCURRENCY at DCL command level prior to compilation, the quoted character string to which you define it will be the currency string. To do this, prior to compiling the COBOL program, issue the following DCL command:

   $ DEFINE SYSSCURRENCY "quoted-character-string"

   The COBOL compiler will utilize the first character of this string as the currency symbol for the program.

   Subsequently, the system default value of SYSSCURRENCY can be restored for the process with the following DCL command:

   $ DEASSIGN SYSSCURRENCY

   The default currency sign can also be established based on the nationality compiler option, depending on the keyword, as follows:
The default currency sign and symbol are the dollar sign ($), and Japanese language support features are disabled.

The default currency sign and symbol are the Yen sign (¥) (which is not overridden by a SYS$CURRENCY definition), and Japanese language support features are enabled, including national character user-defined-words, data items (PIC N), and literals (N”

---

**CURRENCY SIGN Clause (Alpha, I64)**

1. To use CURRENCY SIGN IS literal-7, you must compile the program with the /RESERVED_WORDS=200X qualifier. Without that qualifier, you can specify only CURRENCY SIGN IS char, and specify it only once.

2. The CURRENCY SIGN IS literal-7 clause specifies a currency string that is placed into numeric-edited data items when they are used as receiving items and de-edited from a data item when the data item is used as a sending item that has a numeric or numeric-edited receiving item. The clause also determines which symbol shall be used in a picture character string to specify the presence of this currency string. This symbol is referred to as the currency symbol.

   literal-7 represents the value of the currency string.

   If the CURRENCY SIGN clause is specified with the PICTURE SYMBOL phrase, literal-8 is the currency symbol; if the clause is specified without the PICTURE SYMBOL phrase, literal-7 is the currency symbol, and it must be one character in length.

   If the currency symbol is a lowercase letter, it is treated as its uppercase equivalent.

3. If the PICTURE SYMBOL phrase is not specified, literal-7 must consist of a single character that is not one of the following:
   - 0 through 9
   - A, B, C, D, E, N, P, R, S, V, X, Z, or the lowercase equivalents; or the space
   - Asterisk (*), plus sign (+), minus sign (-), comma (,), period (.), semicolon (;), quotation mark ("), equal sign (=), slash (/), left parenthesis ( ( ), or right parenthesis ( )

4. If the PICTURE SYMBOL phrase is specified, literal-7 can have any length and:
   - Must contain at least one nonspace character, and
   - Can consist of any characters from the computer's character set except for the digits 0 through 9 and the characters asterisk (*), plus sign (+), minus sign (-), comma (,), and period (.)

5. literal-8 can be any character from the computer's character set except for the following:
   - 0 through 9
   - A, B, C, D, E, N, P, R, S, V, X, Z, or the lowercase equivalents; or the space
   - Asterisk (*), plus sign (+), minus sign (-), comma (,), period (.), semicolon (;), quotation mark ("), equal sign (=), slash (/), left parenthesis ( ( ), or right parenthesis ( )

---

**DECIMAL-POINT IS COMMA Clause**

1. The DECIMAL-POINT IS COMMA clause exchanges the functions of the comma and period in: (1) the PICTURE clause character-string and (2) numeric literals.

---

**CURSOR IS Clause (Alpha, I64)**
1. The CURSOR IS clause specifies the initial position of the cursor at the start of an ACCEPT (Format 5) statement. If cursor-position is within an input or update field on the screen, then the initial cursor position is at the start of that field. If the CURSOR IS clause is not specified, or if cursor-position is not within an input or update field on the screen, the cursor’s initial position is at the start of the first input or update field of the screen. The cursor-position is updated upon completion of the ACCEPT statement to contain the position of the cursor when the ACCEPT terminated.

2. In the CURSOR IS clause, if cursor-position is four characters in length, the first two characters represent the line number, and the second two the column number. If cursor-position is six characters in length, the first three characters represent the line number, and the second three the column number.

**CRT STATUS IS Clause (Alpha, I64)**

1. If the CRT STATUS IS clause is specified, crt-status-code is updated after every ACCEPT (Format 5) statement. The first two characters are a termination code that indicates the cause of the termination of the ACCEPT operation. (The third character is currently not defined, and is reserved for future use.) The termination codes are explained in Table 4.1, “CRT STATUS Termination Codes (Alpha, I64)”.

**Command Line Arguments (Alpha, I64)**

2. The ARGUMENT-NUMBER and ARGUMENT-VALUE clauses are used to process command line arguments. The DISPLAY statement is used to select and modify the values, and the ACCEPT statement is used to retrieve the values.

**Environment Variables and System Logicals (Alpha, I64)**

3. The ENVIRONMENT-NAME and ENVIRONMENT-VALUE clauses are used to process environment variables and system logicals. The DISPLAY statement is used to select and modify the values, and the ACCEPT statement is used to retrieve the values.

**Table 4.1. CRT STATUS Termination Codes (Alpha, I64)**

<table>
<thead>
<tr>
<th>First Character</th>
<th>Second Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>’0’</td>
<td>’0’</td>
<td>Terminator key pressed by the operator; normal completion</td>
</tr>
<tr>
<td>’0’</td>
<td>’1’</td>
<td>Auto-skip out of the last field; normal completion</td>
</tr>
<tr>
<td>’1’</td>
<td>x’00’ – x’1A’</td>
<td>User-defined function key number for F1–F20 and the Find through Next keys ¹</td>
</tr>
<tr>
<td>’9’</td>
<td>x’00’</td>
<td>No items falling within the screen ¹</td>
</tr>
</tbody>
</table>

¹The second character contains a hexadecimal value. An example of how to examine this value is given in the Examples section.

**Additional References**

- OBJECT-COMPUTER Paragraph
- CODE-SET Clause
- Section 6.2.6: Scope of Names
- Section 6.5.4: Switch-Status Condition
- ACCEPT Statement
- DISPLAY Statement
- SET Statement
- Appendix B: Character Sets
Examples

1. device-name clause:

   
   ```
   CARD-READER IS THE-CARDS
   CONSOLE IS LOCAL-USER
   ```

   On Tru64 UNIX, this example allows ACCEPT and DISPLAY statements to use THE-CARDS to refer to the environment variable COBOL_CARDREADER and LOCAL-USER to refer to the environment variable COBOL_CONSOLE.

   On OpenVMS, this example allows ACCEPT and DISPLAY statements to use THE-CARDS to refer to the logical name COB$CARDREADER and LOCAL-USER to refer to the logical name COBS_CONSOLE.

2. Top-of-page-name clause:

   
   ```
   C01 IS STARTING-NEW-FORM
   ```

   The following WRITE statement causes the line to appear on the first line of a new page:

   ```
   WRITE REPORT-REC AFTER STARTING-NEW-FORM.
   ```

3. SWITCH clause:

   
   ```
   SWITCH 1 IS FIRST-SWITCH ON IS ONE-ON OFF IS ONE-OFF
   SWITCH-4 ON FOUR-ON
   ```

   (Procedure Division statements can use the condition-names defined in the SWITCH clause. The SET statement can change the status of a switch.)

   The following results assume that switch 1 is on and switch 4 is off:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Truth Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF FOUR-ON</td>
<td>false</td>
</tr>
<tr>
<td>IF ONE-ON</td>
<td>true</td>
</tr>
<tr>
<td>IF NOT ONE-OFF</td>
<td>true</td>
</tr>
<tr>
<td>IF ONE-ON AND NOT FOUR-ON</td>
<td>true</td>
</tr>
</tbody>
</table>

4. ALPHABET clause:

   
   ```
   ALPHABET EB-CONV IS EBCDIC
   ```

   If a file's SELECT clause contains a CODE-SET IS EB-CONV clause, this ALPHABET clause causes translation from EBCDIC to the native character set when the program reads data from the file.

5. User-defined collating sequence:

   
   ```
   ALPHABET ALPH-B IS
   "A" THRU "Z"
   "9" THRU "0"
   " " ALSO "/" ALSO "\"
   ","
   ```

   This ALPHABET clause defines a collating sequence in which uppercase letters are lower than numeric characters. The space, slash (/), and backslash ( \ ) characters have the same position in the collating sequence. The comma is the next higher character. It is implicitly followed by the rest of the character set.

   The following Procedure Division conditional statements show the effect of this ALPHABET clause when the OBJECT-COMPUTER paragraph contains the PROGRAM COLLATING SEQUENCE IS ALPH-B clause:
### 6. User-defined collating sequence with numeric literals:

<table>
<thead>
<tr>
<th>Statements</th>
<th>Truth Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE “A” TO ITEMA.</td>
<td></td>
</tr>
<tr>
<td>MOVE “9” TO ITEMB.</td>
<td></td>
</tr>
<tr>
<td>IF ITEMA &lt; ITEMB</td>
<td>true</td>
</tr>
<tr>
<td>MOVE “ ” TO ITEMA.</td>
<td></td>
</tr>
<tr>
<td>MOVE “\” TO ITEMB.</td>
<td></td>
</tr>
<tr>
<td>IF ITEMA = ITEMB AND ITEMB &gt; “Z”</td>
<td>true</td>
</tr>
<tr>
<td>MOVE “1” TO ITEMA.</td>
<td></td>
</tr>
<tr>
<td>MOVE “9” TO ITEMB.</td>
<td></td>
</tr>
<tr>
<td>IF ITEMA &lt; ITEMB</td>
<td>false</td>
</tr>
</tbody>
</table>

6. User-defined collating sequence with numeric literals:

```
ALPHABET ALPH-C IS 128 THRU 1
```

This clause inverts the positions of the ASCII characters.

The following Procedure Division statements assume that the OBJECT-COMPUTER paragraph contains the SEQUENCE IS ALPH-C clause:

<table>
<thead>
<tr>
<th>Statements</th>
<th>Truth Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE “A” TO ITEMA.</td>
<td></td>
</tr>
<tr>
<td>MOVE “B” TO ITEMB.</td>
<td></td>
</tr>
<tr>
<td>IF ITEMA &lt; ITEMB</td>
<td>false</td>
</tr>
<tr>
<td>MOVE “9” TO ITEMA.</td>
<td></td>
</tr>
<tr>
<td>IF ITEMA &lt; “2”</td>
<td>true</td>
</tr>
<tr>
<td>MOVE “HELLO” TO ITEMA.</td>
<td></td>
</tr>
<tr>
<td>IF ITEMA &gt; SPACES</td>
<td>false</td>
</tr>
</tbody>
</table>

### 7. SYMBOLIC CHARACTERS clause:

```
SYMBOLIC CHARACTERS ESCAPE POUND DOUB-L ARE 28 36 55.
```

The following DISPLAY statement displays the literal “Enter value” in double width on an ANSI terminal.

```
DISPLAY "Enter value" ESCAPE POUND DOUB-L.
```

### 8. CURRENCY SIGN clause:

a. The following example applies to any system, and (if on Alpha or I64) regardless of whether / RESERVED_WORDS=200X is specified when the program is compiled:

```
CURRENCY SIGN "G".
```

```
. 01 ITEMA PIC X(5).
01 ITEMB PIC X(5).
01 ITEMC PIC GG,GG9.99.
01 ITEDM PIC ZZZ.ZZZ99.
01 ITHEME PIC ZZZ,.
```

The following MOVE statements show the effect of the CURRENCY SIGN clause (the character s represents a space):
### Environment Division

<table>
<thead>
<tr>
<th>Statement</th>
<th>ITEMC Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE 12.34 TO ITEMC</td>
<td>sssG12.34</td>
</tr>
<tr>
<td>MOVE 100 TO ITEMC</td>
<td>ssG100.00</td>
</tr>
<tr>
<td>MOVE 1000 TO ITEMC</td>
<td>G1,000.00</td>
</tr>
</tbody>
</table>

b. The following example applies only on Alpha and I64 and only if `/RESERVED_WORDS=200X` is specified when the program is compiled:

CURRENCY SIGN IS "G"
CURRENCY SIGN IS "USD" WITH PICTURE SYMBOL "U"
CURRENCY SIGN IS "DM" WITH PICTURE SYMBOL "D"
CURRENCY SIGN IS "M".

. 01 ITEMA PIC GG,GG9.99.
01 ITEMC PIC DD,DD9.99.
01 ITEMD PIC MMM,MM9.99.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE 12.34 TO ITEMA</td>
<td>ITEMA = sssG12.34</td>
</tr>
<tr>
<td>MOVE 1000 TO ITEMB</td>
<td>ITEMB = USD1,000.00</td>
</tr>
<tr>
<td>MOVE 12.34 TO ITEMC</td>
<td>ITEMC = ssDM12.34</td>
</tr>
<tr>
<td>MOVE 1000 TO ITEMD</td>
<td>ITEMD = sM1,000.00</td>
</tr>
</tbody>
</table>

9. DECIMAL-POINT IS COMMA clause:

<table>
<thead>
<tr>
<th>Statement</th>
<th>ITEMD Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE 1 TO ITEMD</td>
<td>ITEMD = ssssss1.00</td>
</tr>
<tr>
<td>MOVE 1000 TO ITEMD</td>
<td>ITEMD = ss1.000.00</td>
</tr>
<tr>
<td>MOVE 1,1 TO ITEMD</td>
<td>ITEMD = ssssss1,10</td>
</tr>
<tr>
<td>MOVE 12 TO ITEME</td>
<td>ITEME = s12,</td>
</tr>
</tbody>
</table>

10. CURSOR IS clause (Alpha, I64):

SPECIAL- NAMES.
CURSOR IS CURSOR-POSITION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 CURSOR-POSITION.
02 CURSOR-LINE PIC 99.
02 CURSOR-COL PIC 99.

In this example, the cursor's position is defined by data items containing a two-digit line number (CURSOR-LINE) and a two-digit column number (CURSOR-COL).
11. CRT STATUS IS clause (Alpha, I64):

SPECIAL- NAMES.
SYMBOLIC CHARACTERS
FKEY-10-VAL
ARE 11
CRT STATUS IS CRT-STATUS.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 CRT-STATUS.
03 KEY1 PIC 9.
03 KEY2 PIC X.
88 FKEY-10 VALUE FKEY-10-VAL.
03 FILLER PIC X.
.
.
.
ACCEPT MENU-SCREEN.
IF KEY1 EQUAL "0"
PERFORM OPTION_CHOSEN
ELSE IF KEY1 EQUAL "1" AND FKEY-10
DISPLAY "You pressed the F10 key; exiting..." LINE 22.

The first two characters (KEY1 and KEY2) constitute the code that shows the cause of termination of an
ACCEPT operation. (See Table 4.1, “CRT STATUS Termination Codes (Alpha, I64)”.) Note that the SPECIAL-
NAMES paragraph provides for the capturing of the F10 function key.

INPUT-OUTPUT

The the section called “INPUT-OUTPUT” contains two paragraphs:

• FILE-CONTROL paragraph (see FILE-CONTROL)

• I-O-CONTROL paragraph (see I-O-CONTROL)

The FILE-CONTROL paragraph can contain the following clauses:

• ACCESS MODE clause

• ASSIGN clause

• BLOCK CONTAINS clause

• CODE-SET clause

• LOCK MODE clause (Alpha, I64)

• ORGANIZATION clause

• PADDING CHARACTER clause

• RECORD DELIMITER clause

• RESERVE clause

The I-O-CONTROL paragraph can contain the following clauses:

• APPLY clause
This section first describes the FILE-CONTROL paragraph and its clauses, then it describes the I-O-CONTROL paragraph.

**FILE-CONTROL**

FILE-CONTROL — The *FILE-CONTROL* paragraph declares the program's data files.

**General format**

FILE-CONTROL.

**General Format**
Format 1
Format 5

Note

*Clauses marked with an asterisk (*) can be in either the SELECT clause of the Environment Division or the file description entry of the Data Division. They cannot be in both places for the same file.

[file-name]

is the internal name of a file connector. Each file-name must have a file description (or Sort-Merge File Description) entry in the Data Division. The same file-name cannot appear more than once in the FILE-CONTROL paragraph.

Syntax Rules

All Formats

1. SELECT is optional in the FILE-CONTROL paragraph.

2. If SELECT is used in the FILE-CONTROL paragraph, it must be the first clause. Other clauses may follow it in any order.

3. Each file described in the Data Division must be specified only once in the FILE-CONTROL paragraph.

4. On OpenVMS for every format, the first form of ASSIGN TO (marked "OpenVMS ONLY") is available only on the OpenVMS Alpha and OpenVMS I64 operating systems and only if the default /STANDARD=NOXOPEN qualifier is in effect.

   The second form of ASSIGN TO is available on the OpenVMS Alpha and OpenVMS I64 systems if the /STANDARD=XOPEN qualifier is in effect.
Format 6—Report Files

1. Each SELECT clause specifying a Report File must have a file description entry containing a REPORT clause in the Data Division of the same program.

General Rules

Formats 1, 2, 3, and 4—Sequential, Line Sequential, Relative, or Indexed Files

1. You must specify an OPTIONAL phrase for files opened in INPUT, I-O, or EXTEND mode that need not be present when the program runs.

2. The rules for the OPEN statement describe the effects of the OPTIONAL phrase.

3. If the file connector referenced by file-name is an external file connector, all file control entries in the run unit that reference this file connector must have the following characteristics:

   - The same specification for the OPTIONAL phrase
   - A consistent full-file-name
   - The same values for reserve-num, smallest-block, and blocksize
   - The same organization

Format 6—Report Files

1. If the file connector referenced by file-name is an external file connector, all file control entries in the run unit that reference this file connector must have the following characteristics:

   - A consistent full-file-name
   - The same value for reserve-num
   - Sequential organization
   - The same CODE-SET clause

Additional References

- OPEN statement in Chapter 6: Procedure Division
- BLOCK CONTAINS and CODE-SET clauses in this chapter
- FILE STATUS, ACCESS MODE, RECORD KEY, and ALTERNATE RECORD KEY clauses in Chapter 5: Data Division

Examples

The following examples assume that the VALUE OF ID clause is not in any associated file description entry.

1. Sequential file:

   SELECT FILE-A
   ASSIGN TO "INFILE".
This example refers to a file with sequential organization. The word INFILE is equivalent to the nonnumeric literal “INFILE”. If there is no VALUE OF ID clause, the program accesses a file named INFILE.DAT on OpenVMS Alpha and I64 systems, or a file named INFILE on Tru64 UNIX systems.

2. Indexed file:

   SELECT OPTIONAL FILE-A
   ASSIGN TO "INFILE"
   ORGANIZATION INDEXED.

   In this example, the SELECT clause specifies that the indexed file need not be present when the program opens it for INPUT, I-O, or EXTEND.

3. Sort or merge file:

   SELECT SORT-FILE
   ASSIGN TO "SDFILE".

4. Report file:

   SELECT SUMMARY-REPORT
   ASSIGN TO "OUTFIL"
   FILE STATUS IS REPORT-ERRORS.

ASSIGN

ASSIGN — The ASSIGN clause associates a file with a partial or complete file specification.
Description

Format 1 (OpenVMS)
Format 2 (Alpha, I64)
Format 3 (Tru64 UNIX)

[file-spec]

on OpenVMS is either a nonnumeric literal or a COBOL word formed according to the rules for user-defined names. It represents a partial or complete file specification. It must conform to the rules for file specifications as defined by RMS.

[data-name]

is the name of a COBOL data item that contains a partial or complete file specification.

[literal]

is a nonnumeric literal containing a partial or complete file specification.

[DISK (Alpha, I64)]

uses the file specification declared in the optional VALUE OF ID clause as the file name. If the VALUE OF ID clause is not present, file-name-1 is used as the file name in the current directory.

[PRINTER (Alpha, I64)]

creates a print file as if the PRINT-CONTROL phrase of the APPLY clause were specified in the I-O CONTROL paragraph. A print file should contain only printable characters and line and page advancing information written using the ADVANCING clause of the WRITE verb.

[REEL or UNIT (Tru64 UNIX)]

creates the file on a magnetic tape using the ANSI standard format as defined by American National Standard X3.27-1978 (Level 3), Magnetic Tape Labels and File Structure for Information Interchange.
Syntax Rules

1. *data-name* cannot be DISK or PRINTER.

2. EXTERNAL and DYNAMIC are allowed for syntax compatibility with other COBOL vendors. They are treated as documentation only.

3. Format 1 is available only on the OpenVMS operating system and only if the default /STANDARD=NOXOPEN qualifier is in effect.

   Format 2 is available on Alpha and I64 if the /STANDARD=XOPEN qualifier is in effect.

   On Tru64 UNIX, format 2 is the default.

General Rules

1. If there is no VALUE OF ID clause in the file description entry, or that clause contains no file specification, the file specification in the ASSIGN clause is the file specification.

2. If there is a file specification in an associated VALUE OF ID clause, the ASSIGN clause contains the default file specification. File specification components in the VALUE OF ID clause override those in the ASSIGN clause.

3. On OpenVMS, if *file-spec* is not a literal, the compiler:
   - Translates hyphens in the COBOL word to underline characters
   - Treats the word as if it were enclosed in quotation marks

4. *file-spec* may contain a logical name.

5. If you specify ASSIGN TO unquoted string, you need not specify this name in the WORKING-STORAGE section. For example:

   ASSIGN TO TEST1

   This assignment would use "TEST1.DAT" on OpenVMS Alpha and I64.

   On Tru64 UNIX systems, you would specify:

   ASSIGN TO "TEST1.DAT"

   or:

   ASSIGN TO TEST1

   ... WORKING-STORAGE SECTION.
   01 TEST1 PIC X(9) VALUE IS "TEST1.DAT".

6. The file specification derived from one or both of the ASSIGN and VALUE OF ID clauses might refer to an environment variable.

7. On Tru64 UNIX systems, "" is not a valid file specification.

8. On all platforms, *file-spec* must conform to the rules of the operating system where the run-time I-O occurs.

   For indexed files, *file-spec* must conform to the rules of the ISAM package being used. Some older versions of ISAM on Tru64 UNIX may have a 10-character maximum for *file-spec* length.

Format 3

For files assigned to magnetic tape using ASSIGN TO REEL clause:
1. If the length of the file name exceeds 17 characters, it is truncated. Any lowercase characters in a file name are uppercased and others outside the ANSI-"a" character set are converted to ‘Z’.

An "a" character is one of the set of the digits 0,1..9, the uppercase letters A,B..Z, and the following special characters:

```
SP
!"
%
&'
()
*
+
-
.
/
:
;
=
>
?
```

2. Magnetic tape files must be ORGANIZATION SEQUENTIAL and either fixed or variable length record format.

**Technical Notes**

- On all platforms, leading and trailing spaces and tabs are removed from file specifications before the OPEN statement executes.

- When a COBOL OPEN statement executes on an OpenVMS system, the RMS facility:
  - Removes spaces and tab characters from the file specification
  - Translates lowercase letters in the file specification to uppercase
  - Performs logical name translation

  - .DAT is the default file type if one is not specified on an OpenVMS system.

- On Tru64 UNIX, the suffixes added to indexed file names on a Tru64 UNIX system are .idx and .dat.

- On Tru64 UNIX, file specifications are case sensitive.

- Embedded spaces are allowed in file specifications on Tru64 UNIX systems. Thus "file name a" and "Monthly#####Report" are valid file specifications.

- When a COBOL OPEN statement executes on a Tru64 UNIX system, VSI COBOL attempts to match the file specification against an environment variable with the same spelling declared in the current login environment. If an exact match is found, the value of the matching environment variable becomes the file specification. Otherwise, the file specification remains unchanged.

**Additional Reference**

See VALUE OF ID clause in Chapter 5: *Data Division*. For information on defining a file connector, refer to the VSI COBOL User Manual.
**BLOCK CONTAINS**

BLOCK CONTAINS — On OpenVMS systems, the BLOCK CONTAINS clause specifies the size of a physical record. On Tru64 UNIX systems, block size for INDEXED organization is for documentation purposes only.

**General format**

```
[smallest-block]
```

is an integer literal. It specifies the minimum physical record size.

```
[blocksize]
```

is an integer literal. It specifies the exact or maximum physical record size.

**Syntax Rule**

The BLOCK CONTAINS clause can be in the file's Data Division file description entry. However, it cannot be in both the SELECT clause and the file description entry for the same file.

**General Rules**

1. The BLOCK CONTAINS clause specifies physical record size.

2. The compiler ignores smallest-block.

3. The RECORDS phrase specifies physical record size in terms of logical records.
   - For a fixed-length record magnetic tape file, each physical record except the last contains blocksize records.
• For a variable-length record magnetic tape file, the compiler computes the physical record size. It equals the size of the largest logical record, plus any overhead bytes, multiplied by \textit{blocksize}.

4. The \texttt{CHARACTERS} phrase specifies physical record size in terms of characters.

   The physical record size is the maximum of: (1) \textit{blocksize} bytes, and (2) the size of the largest logical record; plus any overhead bytes for variable-length records.

5. If there is no \texttt{BLOCK CONTAINS} clause, physical record size assumes a default value.

   The physical record size is the size of the largest record plus any overhead bytes.

6. The size of physical records (in characters) must be a multiple of four. Otherwise, the I/O system rounds up the physical record size to the next multiple of four.

\textbf{CODE-SET}

\texttt{CODE-SET} — The CODE-SET clause specifies the representation of data on external media.

\textbf{General format}

\begin{verbatim}
[alpha-name]
\end{verbatim}

is the name of a character set defined in the SPECIAL-NAMES paragraph. It cannot be described with literals in the ALPHABET clause.

\textbf{Syntax Rules}

1. The CODE-SET clause can be in the file's Data Division file description entry. However, it cannot be in both the SELECT clause and the file description entry for the same file.
2. The CODE-SET clause applies only to files with sequential organization.

General Rules

1. The CODE-SET clause identifies *alpha-name* as the character set used to represent the file data externally.

2. *alpha-name* specifies how to convert character codes in the file to and from native character codes.

3. Code conversion occurs during execution of an input or output operation. Conversion occurs as if the data were USAGE DISPLAY.

4. Successful OPEN statement execution establishes the character set for code conversion. The set used is the one specified by *alpha-name* in the FILE-CONTROL paragraph implied by the OPEN statement.

5. If there is no CODE-SET clause, no character conversion occurs during input-output operations. The native character set is the default.

Additional Reference

See the SPECIAL-NAMES paragraph.

Example

In this example, the CODE-SET clause specifies that the data in INFILE is coded in the EBCDIC character code set as specified by an alphabet named EB. The SPECIAL-NAMES paragraph defines EB as the EBCDIC character set.

```
SPECIAL-NAMES.
  ALPHABET EB IS EBCDIC.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT INFILE ASSIGN TO INFILE
  CODE-SET IS EB.
```

LOCK MODE (Alpha, I64)

LOCK MODE (Alpha, I64) — The LOCK MODE clause specifies a locking technique to use for a file. LOCK MODE is part of the X/Open COBOL standard.
General format

Syntax Rules

1. X/Open standard and VSI standard syntax cannot both be specified for the same file connector. Hence, if LOCK MODE is specified, the ALLOWING, APPLY LOCK-HOLDING, and REGARDLESS phrases cannot be specified for that file.

2. The WITH LOCK ON RECORD clause is for documentation purposes only.

3. The LOCK MODE IS MANUAL clause is not available for sequential or line sequential files.

General Rules

1. When you specify LOCK MODE IS AUTOMATIC or LOCK MODE IS MANUAL, an OPEN statement (without the WITH LOCK phrase) opens the file in shareable mode. The LOCK MODE clause can be overridden by the WITH LOCK phrase of the OPEN statement.

2. When you specify LOCK MODE IS EXCLUSIVE, a successful OPEN statement opens the file in exclusive mode. The OPEN statement cannot override LOCK MODE IS EXCLUSIVE.

3. If you omit the LOCK MODE clause, opening the file causes it to become exclusive, unless you open it for INPUT, in which case the file becomes shareable.

4. When you specify LOCK MODE IS AUTOMATIC 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.

   If you specify LOCK MODE IS MANUAL, a record lock is acquired by the READ WITH LOCK statement.

5. On Tru64 UNIX, the ROLLBACK phrase is used by ACMSxp applications to specify a recoverable file. You must compile with the -tpx option to specify a recoverable file.
6. A file that is opened in exclusive mode cannot be opened by any other access stream.

7. A file that is in shareable mode can be opened by any number of access streams that do not require exclusive use.

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

**ORGANIZATION**

ORGANIZATION — The ORGANIZATION clause specifies a file's logical structure. On Alpha and I64 systems, the ORGANIZATION IS LINE SEQUENTIAL clause specifies a variant of sequential files compatible with the system text editor.

**General Rules**

1. File organization is established when the file is created. It cannot be subsequently changed.

2. If there is no ORGANIZATION clause, the default is SEQUENTIAL.

3. On Alpha and Itanium systems, when LINE SEQUENTIAL organization is specified, the file is treated as consisting of variable length records, with each record containing one line of data. A line is a sequence of characters ending with a record terminator (n or x'0A'). The terminator is not counted in the length of the record.

4. On Alpha and Itanium systems, a file with LINE SEQUENTIAL organization should only contain printable characters and the record terminator.

5. On Alpha and Itanium systems, a file with LINE SEQUENTIAL organization may not be opened in I-O mode.

6. All programs that open an existing file must specify the same organization with which the file was created.

**Note**

On Tru64 UNIX, a third-party product is required for INDEXED run-time support. Refer to the Release Notes for the latest details on how to obtain the INDEXED run-time support.

**PADDING CHARACTER**

PADDING CHARACTER — The PADDING CHARACTER clause specifies the character to be used to pad blocks in sequential files.
Description

[pad-char]

is a one-character nonnumeric literal or the data-name of a one-character data item. The data-name can be qualified.

General Rule

The PADDING CHARACTER clause is for documentation only.

RECORD DELIMITER (OpenVMS)

RECORD DELIMITER (OpenVMS) — The RECORD DELIMITER clause indicates the method of determining the length of a variable record on the external medium. It is for documentation only.
General Format

On OpenVMS, STANDARD-1 is the I/O system (OpenVMS Record Management System [RMS]) default for tape files. It is the method used for determining the length of a variable-length record. This method is specified in the American National Standard X3.27-1978, “Magnetic Tape Labels and File Structure for Information Interchange,” and International Standard 1001 1979, “Magnetic Tape Labels and File Structure for Information Interchange.”

Additional Reference

For OpenVMS systems, refer to the OpenVMS Record Management Services Reference Manual for more information.

RESERVE

RESERVE — The RECORD DELIMITER clause indicates the method of determining the length of a variable record on the external medium. It is for documentation only.
General Format

[reserve-num]

is an integer literal from 1 to 127. It specifies the number of input-output areas for the file.

General Rule

On OpenVMS systems, if there is no RESERVE clause, the number of input-output areas equals the I/O system default.

Technical Note

For OpenVMS systems, two DCL commands change and display the defaults for block count: SET RMS DEFAULT and SHOW RMS DEFAULT. The number of areas is stored in the MBF field of the RAB.

Additional References

Refer to the RMS documentation for field RAB$B_MBF.

I-O-CONTROL

I-O-CONTROL — The I-O-CONTROL paragraph specifies the input-output techniques to use for a file. On Tru64 UNIX systems, a number of the elements in the I-O-CONTROL paragraph are for documentation only. See the Technical Notes for more information.
General Format

[extend-amt]

is an integer from 0 to 65,535. It specifies the number of blocks in each extension of a disk file.

[file-name]

is the internal name of a file connector. Each file-name must have a file description (or Sort-Merge File Description) entry in the Data Division. The same file-name cannot appear more than once in the FILE-CONTROL paragraph.

[preall-amt]

is an integer from 0 to 4,294,967,295. It specifies the number of blocks to initially allocate when the program creates a disk file.

>window-ptrs]

is an integer from 0 to 127. Its value can also be 255. It specifies the number of retrieval pointers in the window that maps the disk file.

[same-area-file]

names a file described in a Data Division file description entry to share storage areas with every other same-area-file.

[rec-count]

is an integer specifying the number of records to process before writing the rerun information.

[clock-count]
is an integer specifying an interval of time to elapse before writing the rerun information.

[condition-name]

names a switch status which, when set, causes the rerun information to be written. The switch is defined in the SPECIAL-NAMES paragraph of Section 4.1 CONFIGURATION Section.

[multiple-file]

is a file described in a Data Division file description. It specifies that the file shares storage on a reel/unit device with other files. No more than 255 files can be specified.

[pos-integer]

is an integer from 1 to 255. It specifies the relative location of a file on a tape that contains multiple files.

**Syntax Rules**

1. The I-O-CONTROL clauses can appear in any order.

2. As the following table shows, each phrase of the APPLY clause can refer only to some file types.

<table>
<thead>
<tr>
<th>Phrase</th>
<th>File Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTENSION</td>
<td>Disk file</td>
</tr>
<tr>
<td>FILL-SIZE</td>
<td>Indexed organization</td>
</tr>
<tr>
<td>LOCK-HOLDING</td>
<td>Disk file</td>
</tr>
<tr>
<td>MASS-INSERT</td>
<td>Indexed organization</td>
</tr>
<tr>
<td>PREALLOCATION</td>
<td>Disk file</td>
</tr>
<tr>
<td>PRINT-CONTROL</td>
<td>Sequential organization</td>
</tr>
<tr>
<td>WINDOW</td>
<td>Disk file</td>
</tr>
</tbody>
</table>

3. More than one APPLY clause can refer to the same file-name.

4. The phrases of the APPLY clause can appear in any order. However, each phrase can be used only once for each file-name.

5. You can specify the LOCK-HOLDING phrase only if you specify the ALLOWING option of the OPEN statement.

6. The RERUN and MULTIPLE FILE clauses cannot refer to a sort or merge file.

7. In the SAME AREA clause, SORT and SORT-MERGE are equivalent.

8. If same-area-file refers to a sort or merge file, you must use the SORT, SORT-MERGE, or RECORD phrase.

9. A program can contain more than one SAME clause. However, the following conditions apply:
   - A same-area-file cannot be in more than one SAME RECORD AREA clause.
   - A same-area-file that refers to a sort or merge file cannot be in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.

   same-area-files cannot have the global or the external attribute if the program specifies the SAME RECORD AREA phrase.

10. Files specified in a MULTIPLE FILE TAPE clause must be sequential.
11. A file cannot be specified in more than one MULTIPLE FILE TAPE clause.

**General Rules**

**APPLY Clause**

1. An APPLY clause remains active for a file-name until the image terminates.

2. If the file connector referenced by file-name is an external file connector, all file control entries in the run unit that reference this file must have the same APPLY clause.

3. The DEFERRED-WRITE phrase causes a physical write operation to occur only when the input-output buffer for file-name is full. If there is no DEFERRED-WRITE phrase, a physical write occurs each time an output statement executes for file-name.

4. The EXTENSION phrase specifies the number of disk blocks to be added each time a file is extended. The I/O system extends a file when it needs more file space to add a record.

   If extend-amt equals zero, the I/O system extends the file by its default value.

5. The FILL-SIZE phrase causes the I/O system to use the fill size specified when an indexed file is created to fill the file's buckets. If there is no FILL-SIZE phrase, the I/O system fills buckets completely. The FILL-SIZE phrase applies only to indexed files.

6. The LOCK-HOLDING phrase declares the VSI standard manual record-locking attribute for a sequential, relative, or indexed file in a file-sharing environment on disk.

   Once a record is manually locked (see the READ, REWRITE, START, and WRITE statements in Chapter 6: Procedure Division), it remains locked until one of the following occurs:

   • An UNLOCK statement executes.
   • A CLOSE statement executes for the subject file.
   • The image terminates.

   Usage of the APPLY LOCK-HOLDING option requires additional syntax for the OPEN, READ, REWRITE, START, and WRITE verbs. Table 4.2, “Required Manual Record-Locking Phrases (VSI Standard)” summarizes the additional syntax required for Procedure Division I/O statements accessing a file possessing the manual record-locking attribute.

7. X/Open standard and VSI standard syntax cannot both be specified for the same file connector. Hence, APPLY LOCK-HOLDING cannot be specified if LOCK MODE was specified for that file in the SELECT statement.

**Table 4.2. Required Manual Record-Locking Phrases (VSI Standard)**

<table>
<thead>
<tr>
<th>I/O Operation</th>
<th>Procedure Division Options Required by the Manual Record-Locking Facility (VSI Standard)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN</td>
<td>ALLOWING …</td>
</tr>
<tr>
<td></td>
<td>X</td>
</tr>
<tr>
<td>READ</td>
<td>X</td>
</tr>
<tr>
<td>REWRITE</td>
<td>X</td>
</tr>
</tbody>
</table>

**Legend:**

X—Required
N/A—Not Applicable
*—If the ALLOWING option is not specified
### Environment Division

| Procedure Division Options Required by the Manual Record-Locking Facility (VSI Standard) |
|---------------------------------|--------------------------------|
| START                           | X                              | X                           |
| WRITE                           | X                              | N/A                         |

Legend:

- **X**—Required
- **N/A**—Not Applicable
- **_*_***—If the ALLOWING option is not specified

8. The MASS-INSERT phrase is for documentation only. It has no effect on program execution.

9.

10.

11.

**SAME AREA Clause**

1. The SAME AREA clause is for documentation only.

**SAME RECORD AREA Clause**

1. The SAME RECORD AREA clause causes two or more files named by `same-area-file` to share the same memory area for the current logical records.

2. If you specify the SAME RECORD AREA clause, more than one `same-area-file` (or all of them) can be open at the same time.

3. Any record in the shared area becomes the current logical record of:
   - Each `same-area-file` of the SAME RECORD AREA clause open in OUTPUT mode
   - The most recently read `same-area-file` of the SAME RECORD AREA clause open in INPUT mode

The logical records start with the same leftmost character position. Thus, the SAME RECORD AREA clause is equivalent to an implicit redefinition of the shared area.

**SAME SORT (SORT-MERGE) AREA Clause**

1. The SAME SORT (SORT-MERGE) AREA clause is for documentation only.

**RERUN Clause**

1. The RERUN clause is for documentation only. It has no effect on program execution.

**MULTIPLE FILE TAPE Clause**

1. The MULTIPLE FILE TAPE clause specifies the location of a file or files on a reel/unit device. The location of the file or files can be specified as a relative location by providing a `multiple-file` series. The specific file location can be specified by the POSITION phrase.

2. The MULTIPLE FILE TAPE clause specifies the location of a file or files when more than one file shares the same physical reel of tape. If the files in the `multiple-file` sequence are listed in consecutive order, the POSITION phrase is not required. If the files in the `multiple-file` sequence are not listed in consecutive order, the position of the file or files (relative to the beginning of the tape) must be specified in the POSITION phrase.
3. Only those *multiple-files* referenced by the program need to be specified in a MULTIPLE FILE TAPE clause.

4. If a file is specified with a POSITION phrase of a MULTIPLE FILE TAPE clause, subsequent files listed in that MULTIPLE FILE TAPE clause which are not specified with a POSITION phrase are assumed to be in the next higher position.

5. Only one file listed in a MULTIPLE FILE TAPE clause sequence can be open at any one time.

6. If, at run-time, the run-time system determines that the files referenced are not located on a reel device, the MULTIPLE FILE TAPE clause is ignored.

**Technical Notes**

- On Tru64 UNIX systems, many elements of the I-O-CONTROL paragraph are for documentation only. They are accepted and ignored by the compiler. These elements are as follows:
  
  DEFERRED-WRITE
  EXTENSION
  FILL-SIZE
  CONTIGUOUS
  CONTIGUOUS-BEST-TRY
  PREALLOCATION
  PRINT-CONTROL
  WINDOW

- [Additional References](#)

- RESERVE clause

- Technical Notes for the DELETE statement in Chapter 6: *Procedure Division*

- OPEN statement in Chapter 6: *Procedure Division*

- READ statement in Chapter 6: *Procedure Division*

- REWRITE statement in Chapter 6: *Procedure Division*

- START statement in Chapter 6: *Procedure Division*

- UNLOCK statement in Chapter 6: *Procedure Division*

- WRITE statement in Chapter 6: *Procedure Division*

Additional, refer to the VSI COBOL User Manual for more information.
Chapter 5. Data Division

This chapter describes the logical and physical concepts that apply to the Data Division. In addition, this chapter presents the general formats for all Data Division entries and clauses, describes their basic elements, and lists rules of use.

The Data Division defines the data processed by your COBOL program in both physical and logical terms. It also specifies whether the data is contained in files, a database, Oracle CDD/Repository, or is developed only for local use in your program.

The File and Report Sections of your program define data contained in files. A file description, sort-merge file description, or report file description entry creates a logical structure, or file connector, that refers to the physical file. It also can contain clauses that define physical file characteristics. A file description or sort-merge file description entry must be associated with at least one record description entry. A record description entry is a set of one or more data description entries, organized in a hierarchical structure which logically defines a set of related data within the file. The data description entries specify all the data used in your program. You logically define the record hierarchy by the level numbers you use for the data description entries (or entry) within the record description entry. Your logical link to a record or to a field in a record is the data-name you assign in a corresponding data description entry. The clauses in a data description entry also specify physical data attributes, such as storage format and initial values.

A report description entry must be associated with a report group description, which specifies both the logical hierarchy of data in the report and the data's physical attributes.

A screen description entry describes a video form or a portion of a video form.

The Working-Storage and Linkage Sections also contain data description entries, which describe characteristics of data developed for use in your program.

The following sections explain in more detail how a COBOL program specifies physical and logical characteristics. Additionally, the following sections describe how record descriptions impose logical structures on data, and how the physical attributes of data affect the way data is stored and manipulated.

5.1. Logical Concepts of Data Storage

Because a record description is a logical, rather than a physical structure, a program can define more than one record description for the same data. However, this redefinition does not mean that the physical data changes in any way. Multiple record descriptions for the same data all apply to one physical data unit on the file medium.

When you refer to a data-name in a Procedure Division statement, you are referring to a logical unit, either a logical record or a logical subset of that record. When your COBOL source statements execute, the logical units to which they refer are mapped to physical units on media. The logical units are then manipulated according to their physical attributes.

The correspondence between a logical record and a physical record is not necessarily a one-to-one correspondence. The term physical record applies to a data unit that is media dependent and defined by the I/O system. On OpenVMS systems, the I/O system is called OpenVMS Record Management Services (RMS). A logical record may correspond to one physical record, either alone or grouped with other logical records. Or, on disk, a logical record may need more than one physical record to contain it.

Several COBOL clauses (in the Environment and Data Divisions) describe the relationships between logical records and physical records. Programs can then access data as logical entities with little regard to the physical data definitions that the I/O system requires.

During program execution, data transfer between the program and a physical record can involve translation if the SELECT clause contains a CODE-SET clause.

5.1.1. Record Description Entries
Logical records do not have to be subdivided; however, they often are. Subdivision can continue for each of the record's parts, allowing progressively more detailed data definition.

The basic subdivision of a record is the elementary data item (or elementary item), which you define by specifying a PICTURE clause (except for COMP-1 or COMP-2). As the term implies, elementary items are never subdivided. A logical record consists of one or more sets of elementary items, or is itself an elementary item.

A group data item (or group item) is a data set within a record that contains other subordinate data items. The lowest-level group item is always a named sequence of one or more elementary items. Group items can combine to form more inclusive group items. Therefore, an elementary item can be subordinate to more than one group item in the record.

Figure 5.1, “Hierarchical Record Structure” represents a personnel record that illustrates how elementary and group items can be related in a record hierarchy. The record contains three group items directly subordinate to the top level: Identification Data, History, and Payroll Data. The first group item, Identification Data, directly contains two elementary items, Name and Job Title, and two other group items, Employee Number and Address. The group item, Employee Number, contains two elementary items: Department Code and Badge Number. The group item, Address, contains four elementary items: Street, City, State, and ZIP Code. The elementary item, City, belongs to three group items. It is subordinate to Address, Identification Data, and Personnel Record. The second group item, History, directly contains three elementary items: Hire Date, Last Promotion Date, and Termination Date. The third group item, Payroll Data, also directly contains two elementary items: Current Salary and Previous Salary.

**Figure 5.1. Hierarchical Record Structure**

<table>
<thead>
<tr>
<th>Personnel Record</th>
<th>record(level group item)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification Data</td>
<td>(group item)</td>
</tr>
<tr>
<td>Employee Number</td>
<td>(group item)</td>
</tr>
<tr>
<td>Department Code</td>
<td>(elementary item)</td>
</tr>
<tr>
<td>Badge Number</td>
<td>(elementary item)</td>
</tr>
<tr>
<td>Name</td>
<td>(elementary item)</td>
</tr>
<tr>
<td>Address</td>
<td>(group item)</td>
</tr>
<tr>
<td>Street</td>
<td>(elementary item)</td>
</tr>
<tr>
<td>City</td>
<td>(elementary item)</td>
</tr>
<tr>
<td>State</td>
<td>(elementary item)</td>
</tr>
<tr>
<td>ZIP Code</td>
<td>(elementary item)</td>
</tr>
<tr>
<td>Job Title</td>
<td>(elementary item)</td>
</tr>
<tr>
<td>History</td>
<td>(group item)</td>
</tr>
<tr>
<td>Hire Date</td>
<td>(elementary item)</td>
</tr>
<tr>
<td>Last Promotion Date</td>
<td>(elementary item)</td>
</tr>
<tr>
<td>Termination Date</td>
<td>(elementary item)</td>
</tr>
<tr>
<td>Payroll Data</td>
<td>(group item)</td>
</tr>
<tr>
<td>Current Salary</td>
<td>(elementary item)</td>
</tr>
<tr>
<td>Previous Salary</td>
<td>(elementary item)</td>
</tr>
</tbody>
</table>

**5.1.2. Level-Numbers**

Record description entries use a system of level-numbers to specify the hierarchical organization of elementary and group items. Level-numbers that specify hierarchical structure can range from 01 to 49.

The record is the most inclusive data item; that is, there is no hierarchical relationship between one record description entry and any other. However, there is a hierarchical relationship between a group item and its subordinate group or elementary items. The level-number for records is 01. Less inclusive data items have greater (although not necessarily consecutive) level-numbers.

All items subordinate to a group item must have level-numbers greater than the group's level-number. In a record description, a group item is delimited by the next level-number that is less than or equal to that group's level number.

Figure 5.2, “Level-Number Record Structure” shows how level-numbers specify hierarchical structure and how the presence of the PICTURE clause defines an elementary item. Although line indentation can make record
descriptions easier to read, it does not affect record structure; only the level-number values specify the hierarchy. The ellipsis (…) indicates that parts of the program line have been omitted.

**Figure 5.2. Level-Number Record Structure**

Three special level-numbers – 66, 77, and 88 – neither specify hierarchical structure nor actually indicate level. Rather, they define special types of data entries:

- Level-number 66 identifies RENAMES items, which regroup other data items. See the RENAMES clause for more information.

- Level-number 77 specifies noncontiguous (elementary) items in the Working-Storage and Linkage Sections. These data items are not subdivisions of other items and cannot be subdivided. For all other purposes, they are identical to level 01 elementary entries.

- Level-number 88 associates condition-names with values of a corresponding data item (the conditional variable).

  See Chapter 1: *Overview of the COBOL Language*, for more information on condition-names.

### 5.1.3. Multiple Record Description Entries for the Same Data

Example 5.1, “Multiple Record Definition Structure” shows a sample file description entry (FD) that contains three record description entries. The three record description entries define three logical templates the program can impose on a record to access data from it.

**Example 5.1. Multiple Record Definition Structure**

```cobol
FD MASTER-FILE.
  01 T1.
    02 T1-ACCOUNT-NO          PIC 9(6).
    02 T1-TRAN-CODE           PIC 99.
    02 T1-NAME                PIC X(13).
    02 T1-BALANCE             PIC 9(5)V99.
    02 REC-TYPE               PIC XX.
  01 T2.
    02 T2-ACCOUNT-NO          PIC 9(6).
```
5.2. Physical Concepts of Data Storage

COBOL programs describe files and data in physical terms for storage on input-output media. The physical description of data includes the following information:

- The mapping and grouping of logical records within the structure of the file storage medium
- The unit used to transfer records to and from your program
- The size and storage format of an elementary data item

The size of a physical record and the way it is recorded depend on the hardware device involved in an input or output operation. For example, tape and disk media store physical records differently. On tape, a physical record is written between interrecord gaps. On disk, a physical record is written in multiple units of a fixed number of bytes, which is determined by the hardware and operating system involved.

On OpenVMS systems, the term used for a physical record differs according to file organization. A physical record in a sequential file is called a block. A physical record in a relative or indexed file is called a bucket. A block or bucket corresponds to the unit used by the I/O system software to transfer records from a file to your program (and vice versa). The number of records (in logical terms) actually transferred by an input-output operation depends on the following:

- The block size specified by the BLOCK CONTAINS clause (tape files only)
- The number of logical records contained in a physical record

The maximum physical record size depends on file organization and device. On OpenVMS systems, the maximum physical record sizes for sequential files on tape devices and for sequential, indexed, and relative files on disk are shown in terms of number of bytes in Table 5.1, “Maximum Physical Record Size for Tape and Disk Devices”.

Table 5.1. Maximum Physical Record Size for Tape and Disk Devices

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Magnetic Tape Devices</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>65,535 bytes</td>
<td>65,024 bytes</td>
</tr>
<tr>
<td>Indexed</td>
<td>N/A</td>
<td>32,234 bytes</td>
</tr>
<tr>
<td>Relative</td>
<td>N/A</td>
<td>32,255 bytes</td>
</tr>
</tbody>
</table>

Note

A compile-time informational diagnostic appears if the physical record size exceeds 65,024 bytes for a sequential file. However, VSI COBOL programs are device-independent. Therefore, a fatal run-time error can also occur if the file is assigned to disk when the program runs.

5.2.1. Categories and Classes of Data

The size and storage format of an elementary data item depend upon what class and category of data it represents and how that data can be used. A data item's PICTURE clause determines its class and category. The item's
PICTURE clause and USAGE clause, in combination, specify its size and storage format. See the PICTURE and USAGE clauses for more information.

When an arithmetic or data-movement statement transfers data into an elementary item, the category of the item affects the way the data is positioned in storage. The COBOL Standard Alignment Rules (see Section 5.2.2, “COBOL Standard Alignment Rules”) specify the relationship between category and positioning.

Depending on the symbols contained in its PICTURE clause, every elementary item belongs to one of the classes and categories of data items shown in Table 5.2, “Classes and Categories of Data Items”. COMP-1, COMP-2, index data items, and index-names do not have PICTURE clauses; the format of these elementary items is specified by the compiler and they belong to the numeric category.

The class of a group item is treated as alphanumeric regardless of the class of elementary items subordinate to it. Therefore, your program statements should not specify a group item when a numeric item is expected or required.

Table 5.2. Classes and Categories of Data Items

<table>
<thead>
<tr>
<th>Level</th>
<th>Class</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alphabetic</td>
<td>Alphabetic</td>
</tr>
<tr>
<td>Elementary</td>
<td>Numeric</td>
<td>Numeric</td>
</tr>
<tr>
<td></td>
<td>Alphanumeric</td>
<td>Numeric Edited</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alphanumeric Edited</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>Group</td>
<td>Alphanumeric</td>
<td>Alphabetic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Numeric</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Numeric Edited</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alphanumeric Edited</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alphanumeric</td>
</tr>
</tbody>
</table>

5.2.2. COBOL Standard Alignment Rules

The COBOL Standard Alignment Rules specify how characters are positioned in an elementary data item. Positioning depends on the item's category:

1. For a numeric receiving data item:
   • The data is aligned by decimal point. It is moved to the receiving character positions with zero fill or truncation, if necessary.
   • When an assumed decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately after its rightmost character. It is then aligned by decimal point as described in the preceding list item.

2. For a numeric edited receiving data item, the data is aligned by decimal point with zero fill or truncation, if necessary. Editing requirements can replace leading zeros with some other symbol.

3. For receiving data items that are alphabetic, alphanumeric edited, or alphanumeric (without editing), the data is aligned at the leftmost character position in the data item, with space fill or truncation to the right, if necessary.
If the JUSTIFIED clause applies to the receiving item, the rules for the JUSTIFIED clause override rule 3. See the JUSTIFIED clause for more information.

### 5.2.3. Additional Alignment Rules for Record Allocation

As stated in Section 5.2.2, “COBOL Standard Alignment Rules”, the COBOL Standard Alignment Rules specify data positioning only within elementary data items. VSI defines additional alignment rules that affect the positioning of:

- Records on the file media
- Group items within a record
- Elementary items within a group item

On Alpha and I64 systems, VSI COBOL offers the option of allocating subordinate record items along performance-optimal boundaries through the use of the alignment compiler option or directives (or the SYNCHRONIZED clause). If you select one of these options, subordinate data items will be aligned automatically along optimal boundaries for their data type. The compiler may have to skip one or more bytes before assigning a location to the next data item. These skipped bytes, called fill bytes, are spaces between one data item and the next. Refer to the VSI COBOL User Manual for information on using alignment compiler options and directives.

If you do not select one of these Alpha- and I64-only alignment options, the VSI COBOL compiler will locate the data item at the next unassigned byte location.

The presence of fill bytes can make a record's structure different from what you might expect. In particular, if a record contains many items requiring alignment, its size can increase significantly. If, unaware of the fill bytes, you tried to move a group item containing fill bytes to a single data item, right-end truncation would occur. You would not have this problem, however, if you moved the record into another identically defined group item. The method the compiler uses to allocate storage ensures that identically described group items have the same structure, even when their subordinate items are aligned on their required boundaries.

Figure 5.3, “Record Alignment Boundaries” shows alignment boundaries for a record. The boundary is the leftmost location of the 1-, 2-, 4-, or 8-byte area. All boundaries are relative to the beginning of the record as byte number 0.

#### Figure 5.3. Record Alignment Boundaries

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 |
| 2-byte | 2-byte | 2-byte | 2-byte | 2-byte | 2-byte | 2-byte | 2-byte | 2-byte | 2-byte | 2-byte | 2-byte |

The VSI COBOL compiler allocates storage for data items within records according to the rules of the major-minor equivalence technique. The major-minor equivalence technique ensures that identically defined group items have the same structure, even when their subordinate items are aligned. Therefore, group moves always produce predictable results. This technique is based on the following two rules:

- **Location Equivalence**—The leftmost location of a group item is the same as the leftmost location of its first subordinate item.
- **Boundary Equivalence**—The VSI COBOL compiler aligns a group item on a boundary that is as large as the largest boundary for any aligned data item within its scope.
Location Equivalence

Location equivalence forces a group (major) item to the same storage location as its first subordinate (minor) item. This forced positioning occurs regardless of the boundary alignment of either the group or subordinate item.

Refer to the VSI COBOL User Manual chapter on aligning binary data for information on how location equivalence allocates storage.

The following example results in the major-minor location format:

```
01 ITEM-A.
   03 ITEM-B.
      05 ITEM-C PIC 9(4) COMP SYNCHRONIZED.
   03 FILLER PIC X.
   03 ITEM-D.
      05 ITEM-E PIC 9(4) COMP SYNCHRONIZED.
   03 ITEM-F PIC X.
```

The following example (omitting SYNCHRONIZED) results in the left-right location format:

```
01 ITEM-A.
   03 ITEM-B.
      05 ITEM-C PIC 9(4) COMP.
   03 FILLER PIC X.
   03 ITEM-D.
      05 ITEM-E PIC 9(4) COMP.
   03 ITEM-F PIC X.
```

Table 5.3, “Comparison of Major-Minor and Left-Right Locations” compares the major-minor technique of storage allocation with the left-to-right technique that assigns locations to a group item before its subsidiary items. Note that major-minor storage allocation adds a fill byte before ITEM-D. This forces location equivalence with ITEM-E, which is explicitly aligned by the SYNCHRONIZED clause.

Table 5.3. Comparison of Major-Minor and Left-Right Locations

<table>
<thead>
<tr>
<th>Data Item</th>
<th>Major-Minor Location</th>
<th>Left-Right Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEM-A</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>##ITEM-B</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>####ITEM-C</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>##FILLER</td>
<td>02</td>
<td>02</td>
</tr>
<tr>
<td>##ITEM-D</td>
<td>04</td>
<td>03</td>
</tr>
<tr>
<td>####ITEM-E</td>
<td>04</td>
<td>03</td>
</tr>
<tr>
<td>##ITEM-F</td>
<td>06</td>
<td>05</td>
</tr>
</tbody>
</table>

The following diagram also shows the storage allocation for the record ITEM-A in Table 5.3, “Comparison of Major-Minor and Left-Right Locations” using both techniques. A hyphen (-) represents fill bytes caused by explicit alignment; an asterisk (*) represents the FILLER data item.

```
<table>
<thead>
<tr>
<th>Major-Minor Location</th>
<th>Left-Right Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 23 45 67</td>
<td>01 23 45</td>
</tr>
<tr>
<td>AAAAA</td>
<td>AAAAA</td>
</tr>
<tr>
<td>BBBBB</td>
<td>BBBB</td>
</tr>
<tr>
<td>CCCCC</td>
<td>CCCCC</td>
</tr>
</tbody>
</table>
```

Regardless of the record allocation technique, an elementary move always produces the expected result. For example:
**Effect on Group Moves**

A group move may produce an unexpected result, as in the following two situations:

- If ITEM-A of the major-minor location format is moved to ITEM-A of the left-right location format, the fill byte of the major-minor location format overlays the first byte of ITEM-E in the left-right location format; then the first byte of ITEM-E in the major-minor location format overlays the second byte of ITEM-E in the left-right location format, and the second byte of ITEM-E in the major-minor location format overlays ITEM-F in the left-right location format. Finally, ITEM-F in the major-minor location format is truncated.

- A different set of unexpected results occurs if a group move is done in the reverse direction. If ITEM-A of the left-right location format is moved to ITEM-A of the major-minor location format, the first byte of ITEM-D of the left-right location format is moved to the fill byte of the major-minor location format. Then the second byte of ITEM-E in the left-right location format is moved to the first byte of ITEM-E in the major-minor location format, and ITEM-F of the left-right location format is moved to the second byte of ITEM-E in the major-minor location format. Finally, ITEM-F is filled with a space because of the padding rule.

**Boundary Equivalence**

Boundary equivalence forces a group item to a boundary determined by the alignment of its subordinate items.

Within a record, a group item aligns on a boundary as large as the forced alignment boundary of any data item that:

- Is subordinate to the group
- Redefines the group
- Is subordinate to a data item that redefines the group

Refer to the VSI COBOL User Manual chapter on alignment for more information about boundary equivalence.

**Figure 5.4. Effect of Boundary and Location Equivalence Rules on Sample Record**

Figure 5.4, “Effect of Boundary and Location Equivalence Rules on Sample Record” shows how the compiler determines the boundary where each item begins when you specify the no-alignment compiler option.

**Figure 5.5. Storage Allocation for Sample Record** graphically represents Figure 5.4, “Effect of Boundary and Location Equivalence Rules on Sample Record”. It shows the result of location and boundary equivalence applied to the description of record ITEM-A. A hyphen (-) indicates fill bytes.
Note the location of ITEM-D. Location equivalence requires only that it have the same location as ITEM-E, its first subordinate item. ITEM-E requires only 2-byte boundary alignment. However, another of ITEM-D’s subordinate items, ITEM-F, contains ITEM-I, which must be aligned on a 4-byte boundary. Therefore, boundary equivalence forces ITEM-D to a 4-byte boundary as well, causing two fill bytes between ITEM-E and ITEM-F.

This example shows how boundary equivalence helps make group moves predictable:

```
01    ITEM-A.
  03    ITEM-B.
  05    ITEM-C    PIC X.
  05    ITEM-D    PIC 9(8) COMP SYNC.
  03    ITEM-E    PIC X.
  03    ITEM-F.
  05    ITEM-G    PIC X.
  05    ITEM-H    PIC 9(8) COMP SYNC.
  03    ITEM-I    PIC XX.
```

The descriptions of ITEM-B and ITEM-F are equivalent. Therefore, you would not expect the following sentence to change the values of ITEM-C and ITEM-D:

MOVE ITEM-B TO ITEM-F
MOVE ITEM-F TO ITEM-B.

Figure 5.6, “Storage Allocation Without and With Boundary Equivalence” shows how storage for the record would be allocated without and with boundary equivalence. A hyphen (-) indicates fill bytes caused by the SYNCHRONIZED clause. A plus sign (+) represents fill bytes resulting from boundary equivalence.

**Examples**

The following series of examples show major-minor storage allocation. The notes after each example indicate its significant features. A hyphen (-) represents fill bytes.
Example 1

WORKING-STORAGE SECTION.
01 ITEM-A.
  03 ITEM-B PIC X.
  03 ITEM-C.
    05 ITEM-D.
      07 ITEM-E PIC 999 COMP SYNC.
      07 ITEM-F PIC X(10).
    05 ITEM-G REDEFINES ITEM-D.
      07 ITEM-H PIC 9(14) COMP SYNC.
      07 ITEM-I PIC XXXX.
  01 ITEM-J.
    03 ITEM-K.
    05 ITEM-L PIC 999 COMP SYNC.
    05 ITEM-M PIC X(10).
    03 ITEM-N REDEFINES ITEM-K.
    05 ITEM-O PIC 9(14) COMP SYNC.
    05 ITEM-P PIC XXXX.

In this example:
• The relative locations of records (01-level items) in the Working-Storage and Linkage Sections are neither defined nor predictable.
• The structures of ITEM-J (a record) and ITEM-C (a group item within a record) are identical.

Example 2

WORKING-STORAGE SECTION.
01 ITEM-A.
  03 ITEM-B PIC X.
  03 ITEM-C.
    05 ITEM-D OCCURS 3 TIMES.
      07 ITEM-E PIC X.
      07 ITEM-F PIC 9999 COMP SYNC.
      07 ITEM-G PIC X.
    03 ITEM-H PIC X.

In this example:
Data Division

- A fill byte is added after each occurrence of ITEM-D to maintain 2-byte boundary alignment of the next occurrence.
- ITEM-D is 5 bytes long. The fill byte following ITEM-D is not included in its length.
- ITEM-C is 18 bytes long. Its length includes the fill bytes associated with its subordinate items.
- The record ITEM-A is 21 bytes long.

**Example 3**

WORKING-STORAGE SECTION.

```
01 ITEM-A.
  03 ITEM-B PIC X.
  03 ITEM-C.
    05 ITEM-D OCCURS 3 TIMES.
      07 ITEM-E PIC X.
      07 ITEM-F PIC 9999 COMP SYNC.
  03 ITEM-H PIC X.
```

In this example:
- ITEM-G is omitted.
- ITEM-D is 4 bytes long. No fill bytes are added, since the next occurrence is already aligned on a 2-byte boundary.
- ITEM-C is 12 bytes long.
- The record ITEM-A is 15 bytes long.

### 5.2.4. Alpha and I64 Alignment and Padding

The VSI OpenVMS Calling Standard for the Tru64 UNIX, OpenVMS Alpha, and OpenVMS I64 systems specify Alpha natural data alignment and padding. You invoke this alignment by adding the `alignment padding` compiler option to the compile command line, or by using `pad align` directives in your source code. (Refer to the VSI COBOL User Manual for additional information on the command.)

The Alpha natural alignments and field sizes that apply to elementary COBOL data fields are shown in Table 5.4, “Alpha Alignment and Padding”.

#### Table 5.4. Alpha Alignment and Padding

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Alignment Starting Position</th>
<th>Pictures</th>
<th>Usages</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit character string</td>
<td>Byte boundary</td>
<td>A, X, 9, Edited</td>
<td>Display</td>
</tr>
<tr>
<td>16-bit integer</td>
<td>Word boundary, multiple of 2</td>
<td>9(4)</td>
<td>COMP</td>
</tr>
</tbody>
</table>
### 5.3. DATA DIVISION General Format and Rules

#### Function

The Data Division describes data the program creates, receives as input, manipulates, and produces as output.

#### Syntax Rules

1. The Data Division follows the Environment Division.

2. The reserved words DATA DIVISION, followed by a period (.) separator character identify and begin the Data Division.

#### General Rules

1. The Data Division is subdivided into sections. These sections must be in the following order:
File Section

1. The File Section defines the structure of data files. It begins with the File Section header containing the reserved words FILE SECTION, followed by a period (.) separator character.

2. File description entries and sort-merge file description entries follow the File Section header. They can be in any order.

3. The file description entry consists of a level indicator (FD), a file-name, and a series of independent clauses.

4. The FD clause specifies the following:
   - How the file records data
   - The sizes of logical and physical records
   - The names of data records (except for report files)
   - The number of lines on a logical printer page

5. An FD entry, followed by one or more record description entries, defines a sequential, relative, or indexed file. Record description entries must immediately follow the associated FD entry.

6. No record description entries may follow the report file description entry.

7. The sort-merge file description entry consists of a level indicator (SD), a file-name, and a series of independent clauses.

8. An SD clause specifies the following:
   - How the file records data
   - The sizes of logical and physical records
   - The names of data records

9. An SD entry specifies the sizes and names of data records for a sort-merge file referred to by SORT and MERGE statements.

10. An SD entry, followed by one or more record description entries, defines a file. Record description entries must immediately follow the associated SD entry.

Working-Storage Section

1. The Working-Storage Section describes records and subordinate data items. These records are not parts of files; rather, the program develops and processes them internally.

2. The Working-Storage Section also describes data items assigned values by the source program.

3. The Working-Storage Section consists of a section header, followed by record description entries.

4. The section header consists of the reserved words WORKING-STORAGE SECTION, followed by a period (.) separator character.
5. A record description entry groups data items that bear a hierarchical relationship to each other. Unrelated data items in the Working-Storage Section can be described as records that are individual elementary items.

6. Record description clauses can be used in the File Section, the Working-Storage Section, or the Linkage Section.

7. The VALUE IS clause can specify the initial value of any item in the Working-Storage Section except index data items (described by the USAGE IS INDEX clause) and index-names (described by the INDEXED BY phrase of the OCCURS clause).

8. If the VALUE IS clause does not specify an initial value, the default initial value for an item depends on the type of data item:

<table>
<thead>
<tr>
<th>Data Item Type</th>
<th>Default Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>Zero</td>
</tr>
<tr>
<td>Index-name</td>
<td>Occurrence number one</td>
</tr>
<tr>
<td>Index data item</td>
<td>Undefined</td>
</tr>
<tr>
<td>Tables</td>
<td>Undefined</td>
</tr>
<tr>
<td>All others</td>
<td>Spaces</td>
</tr>
</tbody>
</table>

Linkage Section

1. The Linkage Section is used only in a called program.

2. The Linkage Section describes data available through the calling program; both the calling and called programs can access this data.

3. To access calling program data items through the Linkage Section, the called program must have a Procedure Division header USING phrase.

4. The structure of the Linkage Section is the same as that of the Working-Storage Section. It consists of a section header followed by record description entries. The section header consists of the reserved words LINKAGE SECTION followed by a period (.) separator character.

Report Section

1. The Report Section defines report files. It begins with the Report Section header: the reserved words REPORT SECTION, followed by a period (.) separator character.

2. Report description entries follow the Report Section header.

3. The report description entry consists of a level indicator (RD), a report name, and a series of independent clauses.

4. An RD clause specifies the following:
   - Identifying characters to be prefixed to each print line in a report
   - The physical structure and organization of a report
   - The name of the report

5. An RD entry, followed by one or more report group description entries, defines a report. Report group description entries must immediately follow the associated report description entry.

6. A report group description entry defines a report group. It specifies the characteristics of a report group and the individual items within a report group.
Screen Section (Alpha, I64)

1. The Screen Section describes a video form and specifies the attributes, behavior, size, and location of each screen item within the video form. The Screen Section is for use with ACCEPT and DISPLAY statements.

Additional References

- VALUE IS clause
- CALL statement in Chapter 6, Procedure Division
- User-defined words in Section 1.2.1: COBOL Words
- REPORT clause
- Refer to the chapter describing alignment in the VSI COBOL User Manual.

FD (File Description)

FD (File Description)—Sequential, Line Sequential (Alpha, I64), Relative, Indexed, and Report File Descriptions — A file description entry describes the physical structure, identification, record names, and names for sequential, line sequential (Alpha, I64), relative, indexed, and report files. It also specifies the internal or external attributes of a file connector and the local or global attributes of a file-name.
Data Division

Description

General Formats
Format 1—Sequential or Line Sequential Files

FD file-name

[ IS EXTERNAL ]
[ IS GLOBAL ]

* [ BLOCK CONTAINS ] [ smallest-block TO ] blocksize { RECORDS CHARACTERS ]

[ RECORD ]
CONTAINS [ shortest-rec TO ] longest-rec CHARACTERS
[ IS VARYING IN SIZE ]
[ FROM shortest-rec ] [ TO longest-rec ] CHARACTERS
[ DEPENDING ON depending-item ]

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

[ VALUE OF ID ] { quoted-literal-string data-name ]

[ DATA ]
[ RECORD ARE ]
[ RECORD IS ] { rec-name } . . .

[ LINEAGE ]
IS page-size LINES
[ WITH FOOTING AT footing-line ]
[ LINES AT TOP top-lines ]
[ LINES AT BOTTOM bottom-lines ]

[ CODE-SET ] IS alpha-name ]

[ ACCESS MODE ] IS SEQUENTIAL ]

[ FILE STATUS ] IS file-stat ]

Format 2—Relative Files

FD file-name

[ IS EXTERNAL ]
[ IS GLOBAL ]

* [ BLOCK CONTAINS ] [ smallest-block TO ] blocksize { RECORDS CHARACTERS ]

[ RECORD ]
CONTAINS [ shortest-rec TO ] longest-rec CHARACTERS
[ IS VARYING IN SIZE ]
[ FROM shortest-rec ] [ TO longest-rec ] CHARACTERS
[ DEPENDING ON depending-item ]

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

[ VALUE OF ID ] { quoted-literal-string data-name ]

[ DATA ]
[ RECORD ARE ]
[ RECORD IS ] { rec-name } . . .
Format 3—Indexed Files

FD file-name

| IS EXTERNAL ]
| IS GLOBAL ]

• [ BLOCK CONTAINS ] smallest-block TO ] blocksize { RECORDS CHARACTERS }

  [ CONTAINS ] shortest-rec TO ] longest-rec CHARACTERS
  IS VARYING IN SIZE
  [ FROM shortest-rec ] [ TO longest-rec ] CHARACTERS
  [ DEPENDING ON depending-item ]

  LABEL { RECORDS ARE RECORD IS } { STANDARD OMITTED }

  VALUE OF ID IS { quoted-literal-string data-name }

  DATA { RECORDS ARE RECORD IS } { rec-name } . . .

  [ ACCESS MODE IS } { SEQUENTIAL RANDOM DYNAMIC ]

  RECORD KEY IS { rec-key seg-key = (seg) . . . } [ WITH DUPLICATES ] [ ASCENDING DESCENDING ]

  ALTERNATE RECORD KEY IS { alt-key seg-key = (seg) . . . } [ WITH DUPLICATES ] [ ASCENDING DESCENDING ] . . .

  [ FILE STATUS IS file-stat ].

Format 4—Report Files

FD file-name

| IS EXTERNAL ]
| IS GLOBAL ]

• [ BLOCK CONTAINS ] smallest-block TO ] blocksize { RECORDS CHARACTERS }

  [ CONTAINS ] shortest-rec TO ] longest-rec CHARACTERS
  IS VARYING IN SIZE
  [ FROM shortest-rec ] [ TO longest-rec ] CHARACTERS
  [ DEPENDING ON depending-item ]

  LABEL { RECORDS ARE RECORD IS } { STANDARD OMITTED }

  VALUE OF ID IS { quoted-literal-string data-name }

  [ ACCESS MODE IS } [ SEQUENTIAL ]

  REPORT IS REPORTS ARE } { report-name } . . .
*Clauses marked with an asterisk (*) can be in either the SELECT clause of the Environment Division or the file description entry of the Data Division. They cannot be in both places for the same file.

Syntax Rules

Formats 1, 2, 3, and 4—All Files

1. The level indicator FD identifies the start of a file description entry. It must precede file-name.
2. The clauses following file-name can appear in any order.
3. A period (.) separator character must terminate a file description entry.
4. On OpenVMS, the file name written to disk (see the ASSIGN clause in Chapter 4, Environment Division) has the file type .DAT added if no other file type is specified in the corresponding file specification.

Format 1—Sequential or Line Sequential (Alpha, I64) Files

1. file-name can refer only to a sequential file.
2. One or more record description entries must follow the file description entry.

Format 2—Relative Files

1. file-name can refer only to a relative file.
2. If a START statement refers to file-name, the file description must include the RELATIVE KEY phrase within the ACCESS MODE clause.
3. One or more record description entries must follow the file description entry.

Format 3—Indexed Files

1. file-name can refer only to an indexed file.
2. On Tru64 UNIX, for information on file-names for indexed files, see the ASSIGN clause in Chapter 4, Environment Division.
3. alt-key cannot have the same leftmost character position as that of rec-key or any other alt-key for the same file.
4. One or more record description entries must follow the file description entry.

Format 4—Report Files

1. file-name can refer only to a report file.
2. No record description entries may follow the file description entry for a report file.
3. Only the CLOSE statement and the OPEN statement with the OUTPUT or EXTEND phrase may reference this file description entry.

General Rules

Formats 1, 2, 3, and 4—All Files

1. A file description entry associates file-name with a file connector.
2. On OpenVMS, if the file description entry contains the EXTERNAL clause, the RMS special registers RMS-STS, RMS-STV, and RMS-FIENAME are external registers.
3. If the file description entry contains the GLOBAL clause, the RMS special registers RMS-STS, RMS-STV, and RMS-Filename are global registers.

**Format 1—Sequential and Line Sequential (Alpha, I64) Files**

1. If the file description entry contains the LINAGE clause and the EXTERNAL clause, the LINAGE-COUNTER special register is an external data item.
2. If the file description entry contains the LINAGE clause and the GLOBAL clause, the special register LINAGE-COUNTER is a global name.

**Format 3—Indexed Files**

1. If the file description entry contains the EXTERNAL clause, the segmented key `seg-key` has the external attribute.
2. If the file description entry contains the GLOBAL clause, the segmented key `seg-key` is a global name.

Refer to the VSI COBOL User Manual for examples of the file description entry formats.

**SD (Sort-Merge File Description)**

SD (Sort-Merge File Description) — A sort-merge file description entry describes a sort or merge file's physical structure, identification, and record names.

**General Format**

```
SD file-name

RECORD { CONTAINS | shortest-rec TO longest-rec CHARACTERS 
          IS VARYING IN SIZE 
          [ FROM shortest-rec | TO longest-rec ] CHARACTERS 
          [ DEPENDING ON depending-item ] 
          [ DATA { RECORDS ARE 
                    RECORD IS } rec-name ]... .
```

**Syntax Rules**

1. The level indicator SD identifies the start of a sort-merge file description. It must precede `file-name`.
2. The clauses following `file-name` can appear in any order.
3. A period (.) separator character must terminate a sort-merge file description entry.
4. One or more record description entries must follow the sort-merge file description entry.

**General Rule**

No input-output statements can refer to a `file-name` in a sort-merge file description.

**Examples**

Refer to the VSI COBOL User Manual for examples of the sort-merge file description entry.
RD (Report Description)

RD (Report Description) — The Report Description names a report, specifies any identifying characters to be prefixed to each print line in the report, and describes the physical structure and organization of that report. It also determines whether a report-name is a local name or global name.

General Format

```
RD report-name
   | IS GLOBAL |
   | CODE report-code |
   { CONTROL IS Controls ARE }
       { control-name | ... | FINAL control-name | ... } |
   PAGE LIMIT IS Limits ARE page-size LINE LINES |
       | HEADING heading-line |
       | FIRST DETAIL first-detail-line |
       | LAST DETAIL last-detail-line |
       | FOOTING footing-line |
```

Syntax Rules

1. report-name must appear in one and only one REPORT clause.

2. The clauses following report-name may appear in any order. report-name is the highest permissible qualifier that can be specified for LINE-COUNTER, PAGE-COUNTER, and all data-names in the Report Section.

General Rules

1. If the Report Description entry contains the GLOBAL clause, report-name and the special registers LINE-COUNTER and PAGE-COUNTER are global names.

2. The reserved word PAGE-COUNTER references a special register that the compiler creates for each report in the Report Section.

3. In the Report Section, a reference to PAGE-COUNTER can appear only in a SOURCE clause. In the Procedure Division, PAGE-COUNTER can be used anywhere an integer data item can appear.

4. If more than one PAGE-COUNTER exists in a program, PAGE-COUNTER must be qualified by a report-name wherever it is referenced in the Procedure Division.

   In the Report Section, an unqualified reference to PAGE-COUNTER is qualified implicitly by the name of the report containing the reference. Whenever the PAGE-COUNTER of a different report is referenced, PAGE-COUNTER must be explicitly qualified by the other report's report-name.

5. The INITIATE statement causes the Report Writer Control System to set the PAGE-COUNTER of the referenced report to one.

6. PAGE-COUNTER is automatically incremented by one each time the Report Writer Control System executes a page advance.

7. Procedure Division statements may alter the contents of PAGE-COUNTER.
8. The reserved word LINE-COUNTER references a special register that the compiler creates for each report in the Report Section.

9. In the Report Section, a reference to LINE-COUNTER can appear only in a SOURCE clause. In the Procedure Division, LINE-COUNTER can be used anywhere a data item with an integer value can appear. However, only the Report Writer Control System can change the contents of LINE-COUNTER.

10. If there is more than one LINE-COUNTER in a program, Procedure Division references to LINE-COUNTER must be qualified by a report-name.

   In the Report Section, an unqualified reference to LINE-COUNTER is qualified implicitly by the name of the report containing the reference. Whenever the LINE-COUNTER of a different report is referenced, LINE-COUNTER must be explicitly qualified by the other report's report-name.

11. The INITIATE statement causes the Report Writer Control System to set the LINE-COUNTER of the referenced report to zero. The Report Writer Control System also automatically resets LINE-COUNTER to zero each time it executes a page advance.

12. The execution of SUPPRESS statements and the processing of nonprintable report groups do not change the value of LINE-COUNTER.

13. At the time each print line is presented, the value of LINE-COUNTER represents the line number on which the print line is presented. After the presentation of the report group, the value of LINE-COUNTER is governed by the Report Writer Presentation Rules and Tables.

Additional References

- LINE NUMBER (Alpha, I64) clause
- REPORT clause
- FD (File Description)
- Appendix D: Report Writer Presentation Rules and Tables

Example

The following is an example of a global Report Description entry:

FILE SECTION.
FD WEEKLY-REPORTS...
   REPORTS ARE PAYROLL-REPORT
      PAYROLL-IRS.
REPORT SECTION.
RD PAYROLL-REPORT
   IS GLOBAL
   CODE "AA"
   CONTROL GRAND-TOT
      SITE-TOT
      DEPT-TOT
      GROUP-TOT
   PAGE LIMITS ARE 60 LINES
      HEADING      2
      FIRST DETAIL 9
      LAST DETAIL 55
      FOOTING      58.
RD PAYROLL-IRS
The previous example uses the CODE clause to flag PAYROLL-REPORT records from other records (see PAYROLL-IRS) included in the same file (WEEKLY-REPORTS). The entry defines four control totals. GRAND-TOT is the most major control total; it will be printed only at the end of the report. SITE-TOT, DEPT-TOT, and GROUP-TOT are major, intermediate, and minor control totals, respectively. These totals are printed whenever the Report Writer Control System (RWCS) processes a control break. The entry also defines a report page with 60 lines. On each page the RWCS is to print PAYROLL-REPORT headings beginning on line 2, detail lines from lines 9 to 55, and footings beginning on line 58.

Data Description

Data Description — A data description entry specifies the characteristics of a data item.

General Formats

Syntax Rules

1. level-number in Format 1 can be any number from 01 to 49, or 77.

2. Data description clauses can appear in any order, with two exceptions:
   
   - The optional data-name or FILLER clause must immediately follow level-number.
   
   - The optional REDEFINES clause must immediately follow the optional data-name or FILLER clause.
3. The EXTERNAL clause can appear in a level 01 or 77 data description entry in the Working-Storage Section.

4. The GLOBAL clause can appear in a level 01 or 77 data description entry in the Working-Storage Section, or in a level 01 data description entry in the File Section.

5. The EXTERNAL and REDEFINES clauses cannot be in the same data description entry.

6. *data-name* must appear in any Format 1 entry that contains the EXTERNAL clause or GLOBAL clause, or in the record descriptions of a file description entry that contains the EXTERNAL or GLOBAL clause.

7. There must be a PICTURE clause for all elementary items except the following:
   - An index data item
   - A COMP-1 or COMP-2 data item
   - The subject of a RENAMES clause
   - A POINTER data item

   In these cases, there must be no PICTURE clause.

8. The words THRU and THROUGH are equivalent.

9. The SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO clauses can appear only in Data Description entries for elementary items.

**General Rules**

1. Each *condition-name* requires a separate Format 3 entry. The level 88 entry associates one or more values, or ranges of values, with *condition-name*.

   All *condition-name* entries for an associated data item (the conditional variable) must immediately follow that item's data description entry.

   Any *condition-name* associated with a global conditional variable is global.

   A *condition-name* can be associated with a data item at any level except:
   - Another *condition-name*
   - A level 66 item
   - A group that contains items with JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY) clauses
   - An index data item

2. Multiple level 01 data description entries subordinate to an FD or SD entry implicitly redefine the same area.

**Report Group Description**

**Function**

The report group description entry specifies the characteristics of a report group and of the individual items within a report group.
Data Division

General Formats

1. **Format 1**
   - level-number [ group-data-name ]
     - **LINE NUMBER IS**
       - line-num [ ON NEXT PAGE ]
       - PLUS line-num-plus
     - **NEXT GROUP IS**
       - next-group-line-num
       - PLUS next-group-line-num-plus
       - NEXT PAGE

2. **Format 2**
   - level-number [ group-data-name ]
     - **LINE NUMBER IS**
       - line-num [ ON NEXT PAGE ]
       - PLUS line-num-plus
     - **USAGE IS** DISPLAY

3. **Format 3**
   - level-number [ group-data-name ]
     - **JUSTIFIED** RIGHT
     - **LINE NUMBER IS**
       - line-num [ ON NEXT PAGE ]
       - PLUS line-num-plus
     - **PICTURE** IS character-string
     - **SIGN IS** LEADING TRAILING SEPARATE CHARACTER
     - **SOURCE IS** source-name
     - **VALUE IS** literal
     - **SUM** sum-name [ UPON detail-report-group-name ]...
     - **RESET ON** control-foot-name
     - **FINAL**
     - **USAGE IS** DISPLAY
Syntax Rules

All Formats

1. The report group description entry can appear only in the Report Section.

2. Except for the group-data-name clause, which when present must immediately follow level-number, the clauses may be in any sequence.

3. The description of a report group may consist of one, two, or three hierarchical levels:
   a. The first entry that describes a report group must be a Format 1 entry.
   b. Both Format 2 and Format 3 entries may be immediately subordinate to a Format 1 entry.
   c. At least one Format 3 entry must be immediately subordinate to a Format 2 entry.
   d. Format 3 entries must define elementary data items.

4. In the Report Section, the USAGE clause is used only to declare the usage of printable items.
   a. If the USAGE clause appears in a Format 1 or Format 2 entry, at least one subordinate entry must define a printable item.
   b. In Format 3, the USAGE clause must define a printable item.

5. An entry containing a LINE NUMBER clause must not have a subordinate entry that also contains a LINE NUMBER clause.

Format 1

1. group-data-name is required only when:
   a. A GENERATE statement references a DETAIL report group.
   b. An UPON phrase of a SUM clause references a DETAIL report group.
   c. A USE BEFORE REPORTING sentence references a DETAIL report group.
   d. The name of a CONTROL FOOTING report group qualifies a reference to a sum counter.

   If specified, group-data-name can be used as a sum counter qualifier and can be referenced only by:
   • GENERATE statements
   • UPON phrases of the SUM clause
   • USE BEFORE REPORTING declaratives

Format 2

1. level-number can be any integer from 02 to 48 inclusive.

2. A Format 2 entry must contain at least one optional clause.

3. In a Format 2 entry, group-data-name is optional. It can only qualify a sum counter reference.

Format 3

1. level-number can be any integer from 02 to 49 inclusive.
2. A GROUP INDICATE clause can appear only in a DETAIL report group.

3. A SUM clause can appear only in a CONTROL FOOTING report group.

4. An entry containing a COLUMN NUMBER clause but no LINE NUMBER clause must be subordinate to an entry containing a LINE NUMBER clause.

5. group-data-name is optional but can be specified in any entry. group-data-name can be referenced only if the entry defines a sum counter.

6. A LINE NUMBER clause must not be the only clause specified. Refer to Syntax Rule 3d.

7. An entry containing a VALUE clause must also have a COLUMN NUMBER clause.

8. A printable item is a data item whose size and content are specified by an elementary report entry.

9. An elementary report entry contains a COLUMN NUMBER clause, a PICTURE clause, and a SOURCE, SUM, or VALUE clause.

10. Figure 5.7, “Format 3 Clause Combinations” shows all permissible clause combinations for Format 3. You read the table from left to right along the selected row.

**Figure 5.7. Format 3 Clause Combinations**

<table>
<thead>
<tr>
<th>PIC</th>
<th>COLUMN</th>
<th>SOURCE</th>
<th>SUM</th>
<th>VALUE</th>
<th>JUST</th>
<th>BLANK WHEN ZERO</th>
<th>GROUP INDICATE</th>
<th>USAGE</th>
<th>SIGN</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>-</td>
<td>-</td>
<td>M</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
<td>-</td>
<td>M</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>M</td>
<td>P</td>
<td>M</td>
<td>-</td>
<td>-</td>
<td>P</td>
<td>-</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>M</td>
<td>P</td>
<td>M</td>
<td>-</td>
<td>-</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
<td>-</td>
<td>M</td>
<td>P</td>
<td>-</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
</tbody>
</table>

Legend:
- M: Mandatory
- P: Permitted but not required
- -: Not permitted

**General Rules**

1. Format 1 is the Report Group entry. The report group is defined by the contents of this entry and all of its subordinate entries.

2. The BLANK WHEN ZERO clause, the JUSTIFIED clause, and the PICTURE clause for Report Writer are the same as those in the Data Description Section.

**Examples**

The VSI COBOL User Manual contains examples of each report group description entry format.

**Screen Description (Alpha, I64)**

Screen Description (Alpha, I64)

**Function**

A screen description entry describes a video form or a portion of a video form and specifies the attributes, behavior, size, and location of screen items within the video form. The screen description entry is referenced in the Procedure Division by the ACCEPT and DISPLAY statements.
General Formats

Format 1 (Group Screen Item)

```
level-number { screen-name
      FILLER
    }

[ BLANK SCREEN ]
[ FOREGROUND-COLOR IS color-num-1 ]
[ BACKGROUND-COLOR IS color-num-2 ]
[ AUTO ]
[ SECURE ]
[ REQUIRED ]

[ USAGE IS DISPLAY ]

[ SIGN IS { LEADING TRAILING } { SEPARATE CHARACTER } ]

[ FULL ].
```

Format 2 (Elementary Screen Item)

```
level-number { screen-name
      FILLER
    }

[ BLANK { LINE SCREEN } ]
[ BELL ]
[ BLINK ]

[ ERASE { EOL FOS } ]

[ HIGHLIGHT ]
[ LOWLIGHT ]
[ REVERSE-VIDEO ]
[ UNDERLINE ]

[ LINE NUMBER IS PLUS { identifier-1 } ]

[ COLUMN NUMBER IS PLUS { identifier-2 } ]

[ FOREGROUND-COLOR IS color-num-1 ]
[ BACKGROUND-COLOR IS color-num-2 ]

VALUE IS literal-1 .
Format 3 (Elementary Screen Item)

```
level-number | screen-name
             | FILLER

BLANK       | LINE
             | SCREEN

BELL        |
BLINK       |

ERASE       | ECX
             | EOX

HIGHLIGHT   |
LOWLIGHT    |

REVERSE VIDEO |

UNDERLINE   |

LINE NUMBER IS PLUS { identifier-1 integer-1 }

COLUMN NUMBER IS PLUS { identifier-2 integer-2 }

FOREGROUND-COLOR IS color-num-1 |
BACKGROUND-COLOR IS color-num-2 |

PICTURE FILE IS picture-string-1

USING identifier-3


FROM identifier-4 literal-1

TO identifier-5


USAGE IS DISPLAY |

BLANK WHEN ZERO |

JUSTIFIED RIGHT |

SIGN IS LEADING TRAILING | SEPARATE CHARACTER |

AUTO |

SECURE |

REQUIRED |

FULL |
```
2. Each elementary screen description entry must contain at least one of the following clauses:

   BELL
   BLANK
   COLUMN
   LINE
   PICTURE
   VALUE

3. If the FOREGROUND-COLOR, BACKGROUND-COLOR, or SIGN clauses are specified in both the group
   screen description entry and the subordinate description entry for a screen item, then the subordinate screen
   description entry's clauses will take effect.

4. screen-name assigns a name to the screen item described in the screen description entry and must conform to
   the rules for user-defined names. If either the optional screen-name or the key word FILLER is specified, it
   must be the first word following the level number in each screen description entry.

5. If both screen-name and FILLER are omitted, the screen item being described is treated as though FILLER had
   been specified, and cannot be referenced in an ACCEPT or DISPLAY statement.

6. Each level 01 item must have a screen name.

7. A screen item can be referenced only in an ACCEPT or DISPLAY statement.

8. color-num-1 and color-num-2 are integers in the range 0–7. color-num-1 and color-num-2 represent specific
   colors as described in Table 5.5, “Color Table”:

<table>
<thead>
<tr>
<th>Color</th>
<th>Color Value</th>
<th>Color</th>
<th>Color Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>0</td>
<td>Red</td>
<td>4</td>
</tr>
<tr>
<td>Blue</td>
<td>1</td>
<td>Magenta</td>
<td>5</td>
</tr>
<tr>
<td>Green</td>
<td>2</td>
<td>Yellow/Brown</td>
<td>6</td>
</tr>
<tr>
<td>Cyan</td>
<td>3</td>
<td>White</td>
<td>7</td>
</tr>
</tbody>
</table>

9. The USING phrase is equivalent to the combination of FROM and TO phrases, each specifying the same
   identifier.

10. identifier-3, identifier-4 and identifier-5 must be defined in the File, Working-Storage, or Linkage Section.

11. identifier-1 and identifier-2 must be described as elementary unsigned numeric integer data items.

12. literal-1 must be a nonnumeric literal.

13. For a description of picture-string-1, see the PICTURE Clause section of this chapter.

**General Rules**

**All Formats**

1. An input screen item is one whose description contains a TO clause.

2. An output screen item is one whose description contains a FROM clause.

3. A literal screen item is one whose description contains a VALUE clause.

4. An update screen item is one whose description contains a USING clause.

5. An input-output screen item is one whose description contains both a FROM phrase and a TO phrase that may or
   may not reference the same identifier. The rules for update screen items also apply to input-output screen items.
6. The LINE and COLUMN clauses should not be specified within a screen description entry in such a way that fields overlap on the screen or fall beyond the screen boundaries.

**Format 1**

1. Format 1 is used for group screen items.

2. All clauses within a group screen description entry are inherited by subordinate screen description entries with the exception of the BLANK SCREEN clause.

3. If the SECURE clause is specified, it applies to each subordinate input screen item.

4. If the AUTO, FULL, or REQUIRED clauses are specified, they apply to each subordinate input and update screen item.

5. If the BACKGROUND-COLOR, FOREGROUND-COLOR, or SIGN clauses are specified, they apply to each subordinate input, output, and update screen item.

**Format 2**

1. Format 2 is used to describe a literal screen item.

**Format 3**

1. Format 3 is used to describe input, output, or update screen items.

**ACCESS MODE**

**ACCESS MODE**

**Function**

The ACCESS MODE clause specifies the order of access for a file's records.

**General Format**

- **General Format**
- **Format 1—Sequential File**
  
  ```
  [ ACCESS MODE IS ] SEQUENTIAL
  ```

- **Format 2—Relative File**
  
  ```
  [ ACCESS MODE IS ]
  ```

  ```
  { SEQUENTIAL | RELATIVE KEY IS rel-key }
  ```

  ```
  { RANDOM | RELATIVE KEY IS rel-key }
  ```

- **Format 3—Indexed File**
  
  ```
  [ ACCESS MODE IS ]
  ```

  ```
  { SEQUENTIAL |
  ```

  ```
  RANDOM |
  ```

  ```
  DYNAMIC }
  ```

[rel-key]

is the file's RELATIVE KEY data item.
Syntax Rules

1. *rel-key* must be the *data-name* of an unsigned integer data item whose description does not contain a PICTURE symbol (P). It can be qualified but cannot be in a record description entry for the same *file-name*.

2. The ACCESS MODE clause can be in the file's SELECT clause. However, it cannot be in both the SELECT clause and file description entry for the same file.

3. If the USING or GIVING phrases of a SORT or MERGE statement contain the name of the file, the ACCESS MODE RANDOM clause cannot be used for the file.

4. If *rel-key* is associated with an external file connector, *rel-key* must reference the same data item in every program in the run unit.

5. If a START statement references a relative file, the program must specify the RELATIVE KEY phrase for that file.

General Rules

All Formats

1. If there is no ACCESS MODE clause, the access mode is sequential.

2. For sequential access, the sequence in which the program accesses the records depends on the organization of the file, as follows:
   - Sequential files—The sequence is the same as that established by the execution of WRITE statements that created or extended the file.
   - Relative files—The sequence is the order of ascending relative record numbers of the file's existing records.
   - Indexed files—The sequence is the sort order (ascending or descending) of record key values in the established Key of Reference.

Formats 2 and 3

1. For random access, the value of *rel-key* (for relative files) or a Record Key data item (for indexed files) indicates the record to be accessed.

2. For dynamic access, the program can access records sequentially and randomly.

Format 2

1. Relative record numbers uniquely identify records in relative files. A record's relative record number identifies its ordinal position in the file. The first record in the file has a relative record number of 1. Subsequent records have consecutively higher relative record numbers.

2. The Relative Key data item associated with the execution of an Input/Output statement is *rel-key* in the file description entry (or SELECT clause) associated with the statement.

ALTERNATE RECORD KEY

Function

The ALTERNATE RECORD KEY clause specifies an alternate access path to indexed file records.
General Format

**General Format**

```
ALTERNATE RECORD KEY IS { alt-key } WITH DUPLICATES 
[ASCENDING ] 
[DESCENDING ]
```

[alt-key]

is the Record Key for the file. It is the data-name of a data item in a record description entry for the file. It can be qualified, but it cannot be a group item that contains a variable-occurrence data item. The data item must be described as one of the following:

- Alphanumeric item
- Alphabetic item
- Group item
- Unsigned numeric display item
- COMP-3 integer
- COMP integer

[seg-key]

is a segmented-key name that represents the concatenation of one or more (up to eight) occurrences of seg.

[seg]

is the data-name of a data item in a record description entry for the file. It can be qualified, but it cannot be a group item that contains a variable-occurrence data item. The data item must be described as one of the following:

- Alphanumeric item
- Alphabetic item
- Group item
- Unsigned numeric display item

**Syntax Rules**

1. The ALTERNATE RECORD KEY clause can be in the file's SELECT clause. However, for the same file, it cannot be in both the SELECT clause and file description entry.

2. alt-key or the segments of seg-key cannot have the same leftmost character position as that of the Prime Record Key data item or any other alt-key or segment of seg-key for the same file.

**General Rules**

1. seg-key is the concatenation of all specified key segments in the order specified.
2. `seg-key` can be referenced only in a READ (Format 3) or START statement.

3. When a program creates an indexed file with one or more ALTERNATE RECORD KEY clauses, each subsequent program referencing this indexed file must:
   - Use the same data description for `alt-key` or the segments of `seg-key`.
   - Define the same relative location in the record as `alt-key` or the segments of `seg-key`.
   - Specify the same number (or less) of ALTERNATE RECORD KEY clauses.

   On Tru64 UNIX systems, you can specify a different number of keys than was specified when the file was created, if the relaxed key check option (`-relax_key_checking`) is used.

   - Maintain the same order of ALTERNATE RECORD KEY clauses.
   - Specify the same order of keys (ASCENDING or DESCENDING) in each ALTERNATE RECORD KEY clause as the order used when the file was created.

4. The DUPLICATES phrase specifies that two or more records in the file can have duplicate values in the same `alt-key` or the segments of `seg-key`. If there is no DUPLICATES phrase, two records cannot have the same value in corresponding Alternate Record Keys.

   On OpenVMS, if the program was compiled with the `/CHECK=DUPLICATE_KEYS` qualifier on the command line, and the duplicate key specification on a file's FD (in other words, specified in the WITH DUPLICATES phrase) does not match that of the actual file, a run-time diagnostic will be issued when an attempt is made to open the file with an OPEN statement.

   The `/CHECK=DUPLICATE_KEYS` qualifier is not supported for remotely accessed files. Duplicate keys, key length, and number of keys are not checked for remote files, that is, files accessed over the network.

   On Tru64 UNIX systems, DUPLICATES must match the specification for DUPLICATES when the file was created, unless the relaxed key check option is used.

5. If a file has more than one record description entry, only one of these record description entries must describe `alt-key` or the segments of `seg-key`. The character positions referenced by `alt-key` or the segments of `seg-key` in that record description are implicitly referenced in all other record description entries for the file.

6. A file can have up to 254 Alternate Record Keys.

7. If the associated file connector is an external file connector, all File Description entries in the run unit that are associated with the file connector must define the same data description entry for `alt-key` or the segments of `seg-key`, with the same relative location within the record.

8. Each key can be specified as ASCENDING or DESCENDING (ASCENDING is the default). In an ASCENDING key, lower key values occur toward the beginning of the sorted file. In a DESCENDING key, higher key values occur toward the beginning of the sorted file.

**Additional Reference**

- RECORD KEY clause
- SORT statement in Chapter 6, *Procedure Division*
- MERGE statement in Chapter 6, *Procedure Division*
**Function**

In the context of ACCEPT, the AUTO clause moves the cursor to the next field when the last character of an input or update field that was defined with the AUTO clause is entered.

**General Format**

```
AUTO
```

**Syntax Rule**

The AUTO clause cannot be specified in the description of a literal screen item.

**General Rules**

1. If the AUTO clause is specified at group level, it applies to each input and update screen item in that group.

2. The AUTO clause is significant in the context of an ACCEPT.

3. The AUTO clause is ignored in the description of an output screen item.

4. If there is only one field to input, or if the field is the last one of the screen, the ACCEPT statement is completed when the last character of the field is entered.

**Additional Reference**

ACCEPT statement in Chapter 6, *Procedure Division*

**BACKGROUND-COLOR (Alpha, I64)**

**Function**

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

**General Format**

```
BACKGROUND-COLOR IS color-num
```

**Syntax Rule**

The BACKGROUND-COLOR clause can be specified in any screen description entry.

---

<table>
<thead>
<tr>
<th>Color</th>
<th>Color Value</th>
<th>Color</th>
<th>Color Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>0</td>
<td>Red</td>
<td>4</td>
</tr>
<tr>
<td>Blue</td>
<td>1</td>
<td>Magenta</td>
<td>5</td>
</tr>
<tr>
<td>Green</td>
<td>2</td>
<td>Yellow/Brown</td>
<td>6</td>
</tr>
<tr>
<td>Cyan</td>
<td>3</td>
<td>White</td>
<td>7</td>
</tr>
</tbody>
</table>
General Rules

1. The BACKGROUND-COLOR clause is effective only with color screens.

2. If the BACKGROUND-COLOR clause is omitted, the initial default background color is black.

3. If the clause is specified at group level, it applies to all subordinate screen items.

4. If the BLANK SCREEN clause is specified and the BACKGROUND-COLOR clause is specified or inherited, then when a DISPLAY statement displays the screen item, the specified color becomes the default background color. It remains the default background color until another screen item with this combination of options is displayed (whether in the same DISPLAY statement or in another).

Technical Note

The colors in the list above are supported only on terminals and workstations that support the ANSI Standard color sequences.

Additional References

- ACCEPT statement in Chapter 6, Procedure Division
- DISPLAY statement in Chapter 6, Procedure Division

BELL

Function

The BELL clause sounds the workstation or terminal audio tone.

General Format

BELL

Syntax Rule

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

General Rule

The audio tone sounds when a DISPLAY statement displays a screen item whose description contains a BELL clause.

Additional Reference

DISPLAY statement in Chapter 6, Procedure Division

BLANK

Function

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

Syntax Rules
1. The BLANK SCREEN clause can be specified for any screen description entry.
2. The BLANK LINE clause can be specified only for elementary screen description entries.

General Rules
1. The BLANK SCREEN clause executes before a screen item displays, no matter where it appears in the screen item's description. When the BLANK SCREEN clause is specified, the screen is cleared and the cursor is placed at line 1, column 1.
2. When BLANK LINE is specified in an elementary screen item's description, blanking begins at column 1 of the specified line and continues through to the end of the line.
3. If neither the BLANK clause nor the ERASE clause (Alpha, I64) is specified, only the particular character positions corresponding to the screen item are modified when the item is displayed. The remainder of the screen content is not changed.
4. The BLANK SCREEN clause returns the screen to the initial defaults for background and foreground color if the BACKGROUND-COLOR and FOREGROUND-COLOR clauses are not specified, respectively.
5. The BLANK clause is ignored in an ACCEPT statement.

Additional Reference
DISPLAY statement in Chapter 6, Procedure Division

BLANK WHEN ZERO

Function
The BLANK WHEN ZERO clause replaces zeros with spaces when a data item's value is zero. In the context of the Screen Section, it displays spaces when the value of a screen item to be displayed on the screen is zero.

General Format

Syntax Rules
1. The BLANK WHEN ZERO clause can be used only for a numeric or numeric edited elementary item.
2. A data item or screen item containing the BLANK WHEN ZERO clause must be implicitly or explicitly described with DISPLAY usage.
3. The syntax for a data item allows the spelling ZERO or ZEROES or ZEROS. The syntax for a screen item allows the spelling ZERO only.

**General Rules**

1. The BLANK WHEN ZERO clause causes a data item or screen item to contain spaces when its value is zero.
2. When the data item or screen item has a numeric PICTURE string, the BLANK WHEN ZERO clause makes the item's category numeric edited.
3. The BLANK WHEN ZERO clause is ignored in the description of an input screen item.

**Additional Reference**

DISPLAY statement in Chapter 6, *Procedure Division*

**BLINK (Alpha, I64)**

BLINK (Alpha, I64)

**Function**

The BLINK clause displays characters on the screen with the *blink on* character attribute.

**General Format**

```
BLINK
```

**Syntax Rule**

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

**General Rule**

Blinking is only detectable when any of the following conditions are true:

- Nonspace characters are displayed.
- The underline and/or reverse-video attributes are specified.
- The terminal screen is set to light background.

**Additional References**

- DISPLAY statement in Chapter 6, *Procedure Division*
- ACCEPT statement in Chapter 6, *Procedure Division*

**CODE**

CODE

**Function**

The CODE clause specifies a two-character literal that identifies each print line as belonging to a specific report.

```
CODE report-code
```
Data Division

[report-code]

must be a two-character nonnumeric literal.

**Syntax Rule**

If the CODE clause is specified for any report in a file, it must be specified for all reports in that file.

**General Rules**

1. When the CODE clause is specified, `report-code` is automatically placed in the first two character positions of each Report Writer logical record.

2. The positions occupied by `report-code` are not included in the description of the print line, but are included in the logical record size.

**Additional Reference**

RD (Report Description)

**Example**

The following file contains three reports:

FILE SECTION.
FD REPORT-FILE
  LABEL RECORDS ARE STANDARD
  REPORTS ARE REPORT1
  REPORT2
  REPORT3.
REPORT SECTION.
RD REPORT1...
  CODE "AA".
RD REPORT2...
  CODE "BB".
RD REPORT3...
  CODE "CC".

**COLUMN NUMBER**

**COLUMN NUMBER**

**Function**

In a report group description, the COLUMN NUMBER clause identifies a printable item and specifies the position of the item on a print line. In a screen description, the COLUMN NUMBER clause specifies the horizontal screen coordinate for a screen item.

**General Formats**

**Format 1 (Report Description)**

\[\text{COLUMN NUMBER IS column-num}\]

**Format 2 (Screen Description)**

\[\text{COLUMN NUMBER IS [ PLUS \{ identifier-1 \}]}\]
Data Division

[column-num]

is a positive integer greater than zero.

[identifier-1]

is an elementary unsigned numeric integer data item. It cannot be subscripted.

[integer-1]

is an unsigned integer value.

Syntax Rules (Report Description)

1. The COLUMN NUMBER clause can be specified only at the elementary level within a report group. The COLUMN NUMBER clause, if present, must appear in a Format 3 Report Group Description entry, or be subordinate to an entry that contains a LINE NUMBER clause in a Format 2 Report Group Description entry.

2. A printable item is a data item whose size and content is specified by an elementary report entry.

3. An elementary report entry contains a COLUMN NUMBER clause, a PICTURE clause, and a SOURCE, SUM, or VALUE clause.

4. Each printable item within a given print line must be defined in ascending column number order such that each printable item occupies a unique sequence of contiguous character positions.

Syntax Rules (Screen Description)

1. The COLUMN clause can be specified only in an elementary screen description entry.

2. identifier-1 cannot be subscripted.

General Rules (Report Description)

1. The presence of a COLUMN NUMBER clause indicates that these items, if present, are to be presented on the print line:
   - The object of a SOURCE clause
   - The object of a VALUE clause
   - The sum counter in a SUM clause

   The absence of a COLUMN NUMBER clause indicates that the entry is not printable.

2. Column number 1 is the leftmost position of the print line.

3. column-num specifies the column number of the leftmost character position of the printable item.

4. The Report Writer Control System supplies space characters for all positions of a print line not occupied by printable items.

General Rules (Screen Description)

1. The COLUMN clause, in conjunction with the LINE clause, establishes the starting position for a screen item. This position is an offset from the starting screen coordinates specified in the ACCEPT or DISPLAY statement. The COLUMN clause specifies the horizontal coordinate.

2. The COLUMN clause without the PLUS phrase specifies the absolute column position of the screen item.

3. The COLUMN clause with the PLUS phrase specifies a column number relative to that at which the preceding item ends, regardless of whether or not the ACCEPT or DISPLAY statement displays the preceding item on the screen.
4. A setting of COLUMN 1 is assumed in screen description entries that specify the LINE clause but omit the COLUMN clause.

5. If both the LINE clause and the COLUMN clause are omitted, the following apply:
   • If no previous elementary screen item is defined, LINE 1 COLUMN 1 is assumed.
   • If a previous screen item is defined, the ending line of that previous item and COLUMN PLUS 1 is assumed. The screen item then starts immediately following the preceding screen item.

Additional References

• Report Group Description
• LINE NUMBER (Alpha, I64) clause
• ACCEPT statement in Chapter 6, Procedure Division
• DISPLAY statement in Chapter 6, Procedure Division

Examples (Report Description)

1. The following is an example of the COLUMN NUMBER clause in a LINE NUMBER clause:

   ```assembly
   02 LINE 10 COLUMN 1 PIC X(11) VALUE "TOTAL ITEMS".
   1 2 3 4
column: 1234567890123456789012345678901234567890
TOTAL ITEMS
   ```

2. The following is an example of the COLUMN NUMBER clause subordinate to a LINE NUMBER clause:

   ```assembly
   02 LINE 5 ON NEXT PAGE.
   03 COLUMN 1 PIC X(10) VALUE "(Id Number".
   03 COLUMN 12 PIC 9999 VALUE 1234.
   03 COLUMN 16 PIC X VALUE ").".
   03 COLUMN 18 PIC X(11) VALUE "TOTAL SALES".
   03 TSAL COLUMN 30 PIC $$$.99- VALUE 123456.78.
   ```

   ```assembly
column: 123456789012345678901234567890123456789012345678901234567890
(Id Number 1234) TOTAL SALES $123,456.78
   ```

CONTROL

Function

The CONTROL clause establishes the levels of the control hierarchy for the report.

General Format

```
{ CONTROL IS
{ CONTROLS ARE }
{ [ name ]... }
{ [ name ]... }
```

[control-name]
is any data-name in the Subschema, File, Working-Storage, or Linkage Section.

**Syntax Rules**

1. control-name can be qualified.

2. Each occurrence of control-name must identify a different data item.

3. control-name must not have a variable-occurrence data item subordinate to it.

4. If the associated report file connector is an external file connector, control-name must reference the same external data item in all programs in the run unit.

**General Rules**

1. The word FINAL specifies the most major control item. From here, the hierarchy descends to control-name, which is the major control; to the next recurrence of control-name, which is an intermediate control; and so forth to the last recurrence of control-name, which is the minor control.

2. A control break is a change in the value of a control-name.

3. FINAL is used when the most inclusive control group in the report is not associated with a control-name.

4. The first time a GENERATE statement is executed, the Report Writer Control System (RWCS) saves the values of all control data items associated with that report. After that, every time a GENERATE statement is executed, the RWCS tests those control data items to see if their values have changed. If so, a control break occurs. This control break is associated with the highest level control item whose value has changed.

5. Control breaks cause the RWCS to present appropriate CONTROL HEADER and CONTROL FOOTING report groups for printing. Figure 5.8, “Control Break Levels and Their Printed Report Groups” shows the report groups the RWCS processes (X) when you define FINAL, major, intermediate, or minor control-name in a CONTROL HEADING or CONTROL FOOTING phrase in a Report Group Description entry. For example, if the value in a major control-name changes, the RWCS processes all major, intermediate, and minor control groups specified in CONTROL HEADING and CONTROL FOOTING report groups.

**Figure 5.8. Control Break Levels and Their Printed Report Groups**

<table>
<thead>
<tr>
<th>Control Break Level</th>
<th>CONTROL HEADING</th>
<th>CONTROL FOOTING</th>
</tr>
</thead>
<tbody>
<tr>
<td>FINAL</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Major</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Intermediate</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Minor</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

6. The RWCS tests for a control break by comparing the contents of each control data item with the prior contents of each control data item that were saved when the previous GENERATE statement for the same report was executed. The RWCS applies the inequality relation test as follows:

- If the control data item is a numeric data item, the relation test is for the comparison of two numeric operands.
- If the control data item is an index data item, the relation test is for the comparison of two index data items.
- If the control data item is other than as described in Figure 5.8, “Control Break Levels and Their Printed Report Groups”, the relation test is for the comparison of two nonnumeric operands.

**Additional References**

- Report Group Description
Examples

1. This example prints a total record count from TOTAL-LINE at the end of the report because control is FINAL. It is a major control break and prints only once.

   WORKING-STORAGE SECTION.
   01 RECORD-COUNT PIC 9(9) VALUE 0.

   REPORT SECTION.
   AD MASTER-REPORT...
   CONTROL IS FINAL.

   01 DETAIL-LINE TYPE IS DETAIL...

   01 TOTAL-LINE TYPE IS CONTROL FOOTING FINAL.
   02 COLUMN 20 PIC X(17) VALUE "TOTAL RECORDS: ".
   02 COLUMN 40 PIC ZZZ,ZZZ,ZZ9 SOURCE RECORD-COUNT.

   PROCEDURE DIVISION.
   BEGIN.
   OPEN INPUT...
   OPEN OUTPUT...
   INITIATE MASTER-REPORT.

   010-READ-FILE.
   READ... AT END GO TO 999-EOJ.
   GENERATE DETAIL-LINE.
   ADD 1 TO RECORD-COUNT.
   GO TO 010-READ-FILE.

   999-EOJ.
   TERMINATE MASTER-REPORT.
   CLOSE...
   STOP RUN.

2. In the following example, a report defines four control totals in the control clause. The source of these control totals is in an input file—INPUT-FILE. The file is presorted in ascending sequence by MAJOR-CONTROL, INTERMEDIATE-CONTROL, and MINOR-CONTROL. The RWCS will monitor these fields in the input file for any changes. If a new record contains data different from the previous record read, the RWCS triggers a control break.

   In this example, if the value in MINOR-CONTROL changes, a break occurs and the RWCS processes the minor control report group CONTROL FOOTING MINOR-CONTROL. If the value in INTERMEDIATE-CONTROL changes, a break occurs and the RWCS processes the intermediate and minor control report groups CONTROL FOOTING INTERMEDIATE-CONTROL and CONTROL FOOTING MINOR-CONTROL. If the value in MAJOR-CONTROL changes, a break occurs and the RWCS processes the major, intermediate, and minor control report groups CONTROL FOOTING MAJOR-CONTROL, CONTROL FOOTING INTERMEDIATE-CONTROL, and CONTROL FOOTING MINOR-CONTROL.

   FILE SECTION.

   FD INPUT-FILE...
   01 INPUT-RECORD.
   02 MAJOR-CONTROL PIC...
   02 ...
   02 MINOR-CONTROL PIC...
   02 ...
   02 INTERMEDIATE-CONTROL PIC...
   02 ...
Data Division

FD      REPORT-FILE...
       REPORT IS SUMMARY-REPORT.
REPORT SECTION.
RD      SUMMARY-REPORT...
       CONTROLS ARE FINAL
       MAJOR-CONTROL
       INTERMEDIATE-CONTROL
       MINOR-CONTROL.

01 DETAIL-LINE TYPE IS DETAIL...
01 TYPE IS CONTROL FOOTING FINAL ...
01 TYPE IS CONTROL FOOTING MINOR-CONTROL...
01 TYPE IS CONTROL FOOTING MAJOR-CONTROL...
01 TYPE IS CONTROL FOOTING INTERMEDIATE-CONTROL...

Data-Name

Data-Name

Function

data-name specifies a data item that your program can explicitly reference. FILLER specifies an item that cannot be explicitly referenced.

General Format
[
   data-name
]
   [ FILLER ]

[data-name]

Syntax Rules

1. In the File, Working-Storage, and Linkage Sections, data-name or the key word FILLER (if present) must be the first word after the level-number in each data description entry.

2. In the Report Section, data-name need not appear in a report group description entry and the key word FILLER must not be used.

General Rules

1. If there is no data-name or FILLER clause, the compiler treats the data item as a FILLER item.

2. The key word FILLER can name a data item. However, a program cannot explicitly refer to FILLER items.

3. The key word FILLER can name a conditional variable. A program cannot refer to the conditional variable. However, it can refer to the value of the conditional variable by referring to its associated condition-names.

4. In the Report Section, data-name must be used when:
   a. data-name represents a report group to be referred to by a GENERATE or a USE statement in the Procedure Division.
   b. Reference will be made to the sum counter in the Procedure Division or Report Section.
   c. The UPON phrase of the SUM clause references a DETAIL report group.
   d. data-name provides sum counter qualification.
5. If this clause is omitted, the Report Writer Control System does not allow explicit references to the data item.

**Examples**

1. Elementary FILLER items:

   In this example, the program can refer only to the group item, ITEMA.

   ```
   01  ITEMA.
   03  FILLER PIC X(10) VALUE SPACES.
   03  PIC X(2) VALUE "AB".
   03  PIC 9 VALUE 6.
   ```

2. Group FILLER items:

   In this example, the program can refer to any elementary item. However, it cannot refer to the record or to the group item that contains ITEMC and ITEMD.

   ```
   01  FILLER.
   03  ITEMA     PIC X(4).
   03  ITEMB     PIC 9(7).
   03  FILLER.
      05  ITEMC PIC X.
      05  ITEMD PIC 9(8)V99.
   03  ITEME     PIC X.
   ```

3. Report Writer items:

   In this Report Writer example, the program can refer to DL-NAME and DETAIL-LINE, but not to the data beginning in LINE 10, COLUMN 25. Note that FILLER cannot be used in place of a Report Writer data-name.

   ```
   01  DETAIL-LINE TYPE IS DETAIL.
   02  LINE 10.
      03  DL-NAME     COLUMN 1   SOURCE INPUT-NAME.
      03              COLUMN 25  SOURCE INPUT-ADDRESS.
   ```

**DATA RECORDS**

**Function**

The DATA RECORDS clause documents the names of a file's record description entries.

**General Format**

```
DATA { RECORD IS RECORDS ARE } { rec-name } ...
```

[rec-name]

is the name of a data record. It must be defined by a level 01 data description entry subordinate to the file description entry.

**Syntax Rule**

The order of appearance of multiple rec-name entries is not significant.
**General Rule**

The DATA RECORDS clause is for documentation only.

**ERASE (Alpha, I64)**

ERASE (Alpha, I64)

**Function**

The ERASE clause clears from the starting cursor position to the end of either the line or the screen.

**Syntax Rule**

The ERASE clause can be specified only for elementary screen description entries.

**General Rules**

1. Blanking begins at the starting position of the screen item in whose description the ERASE EOL clause is included, and continues to the end of the line.

2. Blanking begins at the starting position of the screen item in whose description the ERASE EOS clause is included, and continues through to the end of the screen.

3. If you specify neither the BLANK nor the ERASE clause, only the particular character positions corresponding to the screen item are modified when the element is displayed. The rest of the screen content remains the same.

4. The ERASE clause is ignored in an ACCEPT statement.

**Additional References**

DISPLAY statement in Chapter 6, *Procedure Division*

**EXTERNAL**

EXTERNAL

**Function**

The EXTERNAL clause specifies that a data item or a file connector in a defining program is common to other programs in the run unit if the program defines it identically. The group and elementary data items of an external data record and files associated with an external file connector are available to every program in the image that describes them.

**Syntax Rules**

1. The EXTERNAL clause can appear only in file description entries or in record description entries in the Working-Storage Section.

2. In a record description entry, only level numbers 01 and 77 can specify the EXTERNAL clause.
3. A program and any program it contains, cannot define identical data-names if their data description entries or file description entries have EXTERNAL clauses.

4. The VALUE clause or the REDEFINES clause cannot be in a data description entry that contains or is subordinate to, an entry that contains the EXTERNAL clause.

5. When using the SAME RECORD AREA clause for several files, the Record Description entries or the file description entries for these files must not include the EXTERNAL clause.

6. Entries that contain the EXTERNAL clause must be named.

**General Rules**

1. Data in a record either subordinate to an external FD, or named by the subject of the EXTERNAL clause, is external. Any program in the image that describes and optionally redefines this data may access and process this subject to the following general rules.

2. If two or more programs in the image describe the same external data record, the associated data description entries (except the GLOBAL clause) must be identical. All subordinate data-names, data items, and redefinitions must also be identical.

3. A program that describes an external data record can contain a Data Description entry that redefines the complete external record. This redefinition need not be the same in other programs in the image.

4. Use of the EXTERNAL clause does not imply that the associated file-name or data-name is a global name.

5. The file connector associated with a file description entry is an external file connector.

6. If two or more programs in an image describe the same external file connector, the clauses associated with the description of that file must be functionally identical and any data items referenced by those clauses must be external.

**Technical Notes**

- Each external sequential file becomes a print format file.

- Each external data record becomes a PSECT (on OpenVMS systems) or a global (on Tru64 UNIX systems), whose name is the 01-level record.

  Each file connector for external files becomes a PSECT (on OpenVMS systems) or a global (on Tru64 UNIX systems), whose name is the name of the file. The records associated with this file become external data records.

  External record items and files are implemented as overlayable shared PSECTs (on OpenVMS systems) or globals (on Tru64 UNIX systems). Therefore, the same storage area is shared among all separately compiled programs for that named external record or file. The PSECT or global is created for compatibility with BASIC COMMON/MAP and FORTRAN labeled COMMON.

  On OpenVMS, for more information on overlayable PSECTs, refer to the LINK documentation in the OpenVMS documentation set.

  On Tru64 UNIX, for more information on globals, refer to the ld documentation in the Tru64 UNIX documentation set.

- On Tru64 UNIX systems, an external data item is case-sensitive. By default, an external data item is converted to lowercase for all separately compiled program units. Other programs (VSI COBOL as well as other languages) must specify the data item in lowercase.

  However, if the names option is set to uppercase on the command line, other programs must specify the data item in uppercase. If the names option is set to as_is, the effect on an external data item is as if uppercase were specified. (The as_is setting is used for calling non-COBOL programs with mixed case.)
Additional References

- GLOBAL clause
- REDEFINES clause

Examples

In the following Working-Storage entries, the data items in RECORD-A are available to any program in the run unit that also describes RECORD-A and its data items. RECORD-B and the data items in it are not available to any other program.

01 RECORD-A EXTERNAL.
   03 ITEMA PIC X.
   03 ITEMB PIC X(22).
   03 ITEMC PIC 999.
01 RECORD-B.
   03 ITEMA PIC X(12).
   03 ITEMB PIC X.
   03 ITMEC PIC 9(18).

FILE STATUS

Function

The FILE STATUS clause specifies a data item to contain the status of an input/output operation.

General Format

```plaintext
FILE STATUS IS file-stat
```

[file-stat]

is the data-name of a two-character alphanumeric Working-Storage Section, or Linkage Section data item. `file-stat` is the file's FILE STATUS data item.

Syntax Rules

1. `file-stat` can be qualified.

2. The FILE STATUS clause can be in the file's SELECT clause or in its file description entry. However, it cannot be in both the SELECT clause and the file description entry for the same file.

3. If the FILE STATUS clause is associated with an external file connector, `file-stat` must reference the same data item in all programs in the run unit.

General Rule

After the execution of every I-O statement that refers to the specified file, a value is moved to `file-stat`. This value indicates the file's I-O status after the execution of the I-O statement.

Additional References

- Appendix C: File Status Values
- Section 6.6.8: I-O Status
FOREGROUND-COLOR (Alpha, I64)

**Function**

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

**General Format**

```
FOREGROUND-COLOR IS color-num
```

[color-num] is an integer in the range 0–7 specifying a color as follows:

<table>
<thead>
<tr>
<th>Color</th>
<th>Color Value</th>
<th>Color</th>
<th>Color Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>0</td>
<td>Red</td>
<td>4</td>
</tr>
<tr>
<td>Blue</td>
<td>1</td>
<td>Magenta</td>
<td>5</td>
</tr>
<tr>
<td>Green</td>
<td>2</td>
<td>Yellow/Brown</td>
<td>6</td>
</tr>
<tr>
<td>Cyan</td>
<td>3</td>
<td>White</td>
<td>7</td>
</tr>
</tbody>
</table>

**Syntax Rule**

The FOREGROUND-COLOR clause can be specified in any screen description entry.

**General Rules**

1. The FOREGROUND-COLOR clause is effective only with color screens.

2. If the FOREGROUND-COLOR clause is omitted, the initial default foreground color is white.

3. If the clause is specified at group level, it applies to all subordinate screen items.

4. If the BLANK SCREEN clause is specified and the FOREGROUND-COLOR clause is specified or inherited, then when a DISPLAY statement displays the screen item, the specified color becomes the default foreground color. It remains the default foreground color until another screen item with this combination of options is displayed (whether in the same DISPLAY statement or in another).

5. If the HIGHLIGHT clause is also specified, foreground and background colors are brightened and lightened; for example, black may become grey and brown may become yellow.

**Technical Note**

The colors in the list above are supported only on terminals and workstations that support the ANSI Standard color sequences.

**Additional References**

- ACCEPT in Chapter 6, *Procedure Division*
- DISPLAY in Chapter 6, *Procedure Division*

**FULL (Alpha, I64)**

FULL (Alpha, I64)
Function

The FULL clause specifies that a screen item must be left either completely empty or it must be entirely filled with data.

FULL

Syntax Rules

1. If the FULL clause is specified in a screen description entry, the JUSTIFIED clause cannot be specified.
2. The FULL clause is valid only in the description of an input or update screen item.

General Rules

1. If the FULL clause is specified at group level, it applies to all subordinate input or update screen items.
2. The FULL clause is effective during the execution of any ACCEPT statement when the cursor enters the screen item. Until this clause is satisfied, the operator cannot leave the field and normal terminator keystrokes are rejected.
3. To satisfy the FULL clause for an alphanumeric screen item, either the field must contain all spaces, or both the first and last character positions must contain nonspace characters.
4. To satisfy the FULL clause for a numeric or numeric edited screen item, either the value must be zero or there must be no digit position in which zero suppression has taken effect.
5. 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 by operator-keyed data.
6. The FULL clause is not effective if a function key terminates the accept operation.
7. Specifying the FULL and REQUIRED clauses together requires that the user must always entirely fill the field.
8. The FULL clause is ignored for an elementary output field.

Additional Reference

ACCEPT statement in Chapter 6, Procedure Division

GLOBAL

GLOBAL

Function

The GLOBAL clause specifies that data-name, file-name, or report-name is available to every program contained within the program that declares it.

IS GLOBAL

Syntax Rules

1. The GLOBAL clause can appear only in file description entries, Report Description entries, a data description entry whose level number is 01, in the File or Working-Storage Section, or a data description entry whose level number is 77, in the Working-Storage Section.
2. In the same Data Division, the GLOBAL clause must not appear in Data Description entries that contain identical data-names.
3. If you use the SAME RECORD AREA clause for several files, the Record Description entries or the file
description entries for these files must not include the GLOBAL clause.

4. Entries that contain the GLOBAL clause must be named.

**General Rules**

1. Any data-name, file-name, or report-name specifying the GLOBAL clause is a global name. All data items
   subordinate to a global data-name or file-name are global names. All condition-names associated with a global
   name are global names.

2. A statement in a program contained directly or indirectly within a program that describes a global name may
   reference the name without describing it again.

3. If the GLOBAL clause is used in a data description entry that contains the REDEFINES clause, the global
   attribute applies only to the subject of the REDEFINES clause.

**Technical Note**

Each global sequential file becomes a print format file.

**Additional Reference**

Section 6.2.6: Scope of Names

---

**GROUP INDICATE**

**GROUP INDICATE**

**Function**

The GROUP INDICATE clause specifies that the associated printable item is presented only on the first occurrence
of its DETAIL report group after a control break or page advance.

**GROUP INDICATE**

**Syntax Rule**

The GROUP INDICATE clause must be specified only in a DETAIL report group entry that defines a printable
item.

**General Rules**

1. If the program contains a GROUP INDICATE clause, the compiler suppresses printing of the printable item
   and supplies spaces, except:
   
   a. On the first presentation of the DETAIL report group
   
   b. On the first presentation of the DETAIL report group after every page advance
   
   c. On the first presentation of the DETAIL report group after every control break

2. If the program specifies neither the PAGE clause nor the CONTROL clause in a Report Description entry,
   then the first time a DETAIL report group is presented a GROUP INDICATE printable item is also presented.
   Thereafter, spaces are supplied for indicated items with SOURCE or VALUE clauses.

**Additional Reference**

Appendix D: *Report Writer Presentation Rules and Tables*
Example

The following example shows the effect of the GROUP INDICATE clause on a printable item (SOURCE I-NAME).

<table>
<thead>
<tr>
<th>Sample Program</th>
<th>Without the GROUP INDICATE Clause</th>
<th>With the GROUP INDICATE Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 DETAIL-LINE TYPE IS DETAIL LINE IS PLUS 1.</td>
<td>01 DETAIL-LINE TYPE IS DETAIL LINE IS PLUS 1.</td>
<td>01 DETAIL-LINE TYPE IS DETAIL LINE IS PLUS 1.</td>
</tr>
<tr>
<td>02 COLUMN 1 PIC X(15)</td>
<td>02 COLUMN 1 PIC X(15)</td>
<td>02 COLUMN 1 PIC X(15)</td>
</tr>
<tr>
<td>* GROUP INDICATE</td>
<td>* GROUP INDICATE</td>
<td>* GROUP INDICATE</td>
</tr>
<tr>
<td>SOURCE I-NAME.</td>
<td>SOURCE I-NAME.</td>
<td>SOURCE I-NAME.</td>
</tr>
<tr>
<td>02 COLUMN 20 PIC 9(6)</td>
<td>02 COLUMN 20 PIC 9(6)</td>
<td>02 COLUMN 20 PIC 9(6)</td>
</tr>
<tr>
<td>SOURCE I-INV-NO.</td>
<td>SOURCE I-INV-NO.</td>
<td>SOURCE I-INV-NO.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample Report</th>
<th>Without the GROUP INDICATE Clause</th>
<th>With the GROUP INDICATE Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789012345678901234567890</td>
<td>123456789012345678901234567890</td>
<td>123456789012345678901234567890</td>
</tr>
<tr>
<td>Name</td>
<td>Invoice Number</td>
<td>Name</td>
</tr>
<tr>
<td>Hendrexon B.</td>
<td>123456</td>
<td>Hendrexon B.</td>
</tr>
<tr>
<td>Hendrexon B.</td>
<td>123456</td>
<td>Hendrexon B.</td>
</tr>
<tr>
<td>Hendrexon B.</td>
<td>123456</td>
<td>Hendrexon B.</td>
</tr>
<tr>
<td>Blare R.</td>
<td>123456</td>
<td>Blare R.</td>
</tr>
<tr>
<td>Blare R.</td>
<td>123456</td>
<td>Blare R.</td>
</tr>
<tr>
<td>Blare R.</td>
<td>123456</td>
<td>Blare R.</td>
</tr>
<tr>
<td>Blare R.</td>
<td>123456</td>
<td>Blare R.</td>
</tr>
<tr>
<td>Provinchet R.</td>
<td>123456</td>
<td>Provinchet R.</td>
</tr>
<tr>
<td>Provinchet R.</td>
<td>123456</td>
<td>Provinchet R.</td>
</tr>
<tr>
<td>Provinchet R.</td>
<td>123456</td>
<td>Provinchet R.</td>
</tr>
<tr>
<td>Provinchet R.</td>
<td>123456</td>
<td>Provinchet R.</td>
</tr>
</tbody>
</table>

HIGHLIGHT (Alpha, I64)

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

**HIGHLIGHT**

**Syntax Rule**
The HIGHLIGHT clause can be specified only for an elementary screen description entry.

**Additional References**
- ACCEPT in Chapter 6, *Procedure Division*
- DISPLAY in Chapter 6, *Procedure Division*

**JUSTIFIED**

**Function**
The JUSTIFIED clause specifies nonstandard data positioning in a screen item or another receiving item.

**General Format**
{ JUSTIFIED
  JUST
} RIGHT

**Syntax Rules**
1. The JUSTIFIED clause can be used only for elementary items and alphanumeric data items. It cannot be used for index data items, numeric data items, or edited data items.

2. JUST is the abbreviated form of JUSTIFIED.

**General Rules**
1. If a COBOL statement transfers data to a receiving item whose data description contains the JUSTIFIED clause, the Run-Time System:
   - Truncates the excess leftmost characters if the sending item is larger than the receiving item.
   - Aligns the data at the rightmost character position of the receiving item if the sending item is smaller than the receiving item. (Spaces fill the excess leftmost character positions.)

2. If there is no JUSTIFIED clause, data movement follows the rules for aligning data in elementary items (Standard Alignment Rules).

**Additional References**
- MOVE statement in Chapter 6, *Procedure Division*
- Section 5.2.2: COBOL Standard Alignment Rules

**Examples**
The Procedure Division entry for the MOVE statement contains examples using this clause.

**LABEL RECORDS**

LABEL RECORDS
Function

The LABEL RECORDS clause specifies the presence or absence of labels.

**General Format**

```
LABEL { RECORDS ARE { STANDARD } OMITTED }
```

General Rule

The LABEL RECORDS clause is for documentation only.

Level-Number

**Function**

The `level-number` shows the position of a data item or screen item within the hierarchical structure of a logical record or a report group or a screen description. It also identifies entries for `condition-names` and the RENAMES clause.

**General Format**

```
level-number
```

**Syntax Rules**

1. The `level-number` must be the first element in a data description entry or a screen description entry.

2. Data description entries that are subordinate to a file description (FD) entry have `level-numbers` 01 to 49, 66, or 88.

3. Data description entries in the Working-Storage and Linkage Sections have `level-numbers` 01 to 49, 66, 77, or 88.

4. Report group description entries in the Report Section have `level-numbers` 01 to 49 only. See the Report Group Description entry for additional rules for Report Writer `level-numbers`.

5. Screen description entries in the Screen Section have `level-numbers` 01 to 49 only. See the Screen Description (Alpha, I64) entry for additional rules for Screen Section `level-numbers`.

**General Rules**

1. The `level-number` 01 identifies the first entry in a record description, report group description, or screen description entry.

2. Multiple level 01 entries subordinate to a file description entry represent implicit redefinitions of the same area.

3. Multiple level 01 entries subordinate to a report description entry do not represent implicit redefinitions of the same area.

4. `Level-number` 66 identifies a RENAMES entry. It can be used only in a Format 2 data description entry.

5. `Level-number` 77 identifies a noncontiguous data item entry in the Working-Storage and Linkage Sections. The level 77 entry can have no subordinate data description entries except level 88 items.

6. `Level-number` 88 defines a `condition-name` associated with a conditional variable. It can be used only in a Format 3 data description entry.
7. Level-numbers 66, 77, and 88 do not imply a hierarchical position.

**Additional References**

- RD (Report Description) entry
- Data Description entry
- Report Group Description entry
- RENAMES clause
- Section 1.2.1.1, “User-Defined Words” in Section 1.2.1: COBOL Words
- Section 5.1.1: Record Description Entries
- Screen Description (Alpha, I64) entry

**LINAGE**

**Function**

The LINAGE clause specifies the number of lines on a logical page. It can also specify the size of the logical page's top and bottom margins and the line where the footing area begins in the page body.

**General Format**

```
LINAGE IS page-lines LINES
  | WITH FOOTING AT footing-line |
  | LINES AT TOP top-lines |
  | LINES AT BOTTOM bottom-lines |
```

[page-lines] is a positive integer or the data-name of an elementary unsigned integer numeric data item. Its value must be greater than zero. It specifies the number of lines that can be written or spaced on the logical page. If page-lines is a data-name, it can be qualified.

[footing-line] is a positive integer or the data-name of an elementary unsigned integer numeric data item. Its value must be greater than zero, but cannot be greater than page-lines. footing-line specifies the line number where the footing area begins in the page body. If footing-line is a data-name, it can be qualified.

[top-lines] is an integer or the data-name of an elementary unsigned integer numeric data item. Its value can be zero. top-lines specifies the number of lines in the top margin of the logical page. If top-lines is a data-name, it can be qualified.

[bottom-lines] is an integer or the data-name of an elementary unsigned integer numeric data item. Its value can be zero. bottom-lines specifies the number of lines in the bottom margin of the logical page. If bottom-lines is a data-name, it can be qualified.

**General Rules**

1. The LINAGE clause specifies the number of lines on a logical page.
2. Logical page size is the sum of the values specified in all phrases except FOOTING. If there is no LINES AT TOP or LINES AT BOTTOM phrase, the default value of top-lines or bottom-lines is zero. If there is no FOOTING phrase, the default value of footing-line equals the value of page-lines.

3. Logical and physical page sizes are not necessarily the same.

4. The page body is the logical page area in which the program can write or space lines. Its size equals the value of page-lines.

5. The footing area is the area of the logical page between footing-line and page-lines, inclusive.

6.

7. When page-lines, top-lines, and bottom-lines are data-names, their values affect OPEN and WRITE statement execution as follows:
   - When the program executes an OPEN statement with the OUTPUT phrase for the file, the values specify the number of lines in each of the associated sections of the first logical page.
   - When the program executes a WRITE statement with the ADVANCING PAGE phrase, or when a page overflow condition occurs, the values specify the number of lines in each of the associated sections of the next logical page.

8. The value of footing-line defines the footing area for the first logical page when the program executes an OPEN statement with the OUTPUT phrase for the file. The value defines the footing area for the next logical page when: (a) the program executes a WRITE statement with the ADVANCING PAGE phrase or, (b) a page overflow condition occurs.

9. For each file with a LINAGE clause, the program has a corresponding special register called LINAGE-COUNTER. At any time, the value in LINAGE-COUNTER is the line number in the current page body at which the device is positioned. Other open modes (Input, I-O, and Extend) are not permitted and have unpredictable results.

10. LINAGE-COUNTER is global if a file description entry specifies the GLOBAL clause and the LINAGE clause.

11. LINAGE-COUNTER is a 9-digit numeric special register. Procedure Division statements can refer to LINAGE-COUNTER but cannot change its value.

12. If the program has more than one LINAGE-COUNTER, all Procedure Division references to it must be qualified by file-name.

13. Execution of a WRITE statement for a file with the LINAGE clause changes the value of the associated LINAGE-COUNTER:
   - If the WRITE statement has the ADVANCING PAGE phrase, its execution resets LINAGE-COUNTER to one. (The resetting operation implicitly increments the value of LINAGE-COUNTER to exceed the value of page-lines.)
   - If the WRITE statement has the ADVANCING LINES phrase, its execution increments LINAGE-COUNTER by the value in the ADVANCING phrase.
   - If the WRITE statement does not have the ADVANCING phrase, it increments LINAGE-COUNTER by one.

14.

15. Each logical page follows the preceding logical page with no spacing between them.

16. If the file connector associated with this file description entry is an external file connector, all file description entries in the run unit associated with this file connector must have the following features:
   - A LINAGE clause, if any file description entry has a LINAGE clause
   - The same corresponding integer values for page-lines, footing-lines, top-lines, and bottom-lines
• The same corresponding external data items referenced by page-lines, footing-lines, top-lines, and bottom-lines

Technical Notes

• The /NOVFC compiler option can be used on OpenVMS Alpha and I64 to produce a Stream_LF record-formatted print file. The default (/VFC) behavior is to produce a VFC record-formatted file.

• VSI COBOL on Tru64 UNIX systems writes LINAGE files with blank lines to simulate WRITE ADVANCING behavior. These blank lines would not be produced on an OpenVMS Alpha or I64 system. When you input a LINAGE file, you must compensate for the difference. For example, use an extra initial READ statement (on Tru64 UNIX systems) to skip over the leading blank line in the LINAGE file.

Additional References

• GLOBAL clause

• WRITE statement in Chapter 6, Procedure Division

Example

The following example specifies a logical page whose size is 26 lines:

```cobol
FD PRINT-FILE
  VALUE OF ID IS "REPORT1.LIS"
  LINAGE IS 16 LINES WITH FOOTING AT 13
  LINES AT TOP 4 LINES AT BOTTOM 6.
```

In this example, the first line to which the page can be positioned is the fifth line. The end-of-page condition occurs when a WRITE statement causes the LINAGE-COUNTER value to be in the range 13 to 16. The page overflow condition occurs when a WRITE statement causes the LINAGE-COUNTER value to exceed 16.

Figure 5.9, “Logical Page Areas Resulting from a LINAGE Clause” shows the logical page areas resulting from the example.

Figure 5.9. Logical Page Areas Resulting from a LINAGE Clause

LINE NUMBER (Alpha, I64)

LINE NUMBER (Alpha, I64)
Function

The LINE NUMBER clause specifies vertical positioning information for a report group, or specifies the vertical screen coordinate for a screen item.

General Formats

**Format 1 (Report Description)**

\[
\text{LINE NUMBER IS} \left\{ \begin{array}{c}
\text{line-num [ON NEXT PAGE ]}
\text{PLUS line-num-plus}
\end{array} \right\}
\]

**Format 2 (Screen Description)**

\[
\text{LINE NUMBER IS} \; \text{PLUS} \left\{ \begin{array}{c}
\text{identifier-1}
\end{array} \right\}
\]

[line-num]

is a nonnegative integer. `line-num` represents an absolute line number on a logical page and establishes a print line for a Report Writer report group.

[line-num-plus]

is a positive integer. `line-num-plus` represents a relative line number on a logical page and establishes a print line for a Report Writer report group.

[identifier-1]

is an elementary unsigned numeric integer data item. It cannot be subscripted.

[integer-1]

is an unsigned integer value.

Syntax Rules (Report Description)

1. Neither `line-num` nor `line-num-plus` can exceed three significant digits.

   The PAGE clause defines the length of a logical page and the vertical subdivisions within which each report group is presented. Neither `line-num` nor `line-num-plus` may specify a line outside of the PAGE clause limits. See PAGE clause for more information.

2. Within a given Report Group Description, an entry containing a LINE NUMBER clause must not contain a subordinate entry that also contains a LINE NUMBER clause.

3. Within a given Report Group Description, all absolute LINE NUMBER clauses must precede all relative LINE NUMBER clauses.

4. Within a given Report Group Description, successive absolute LINE NUMBER clauses must specify integers in ascending order. The integers need not be consecutive.

5. If a given Report Description (RD) does not contain a PAGE clause, the program may specify only relative LINE NUMBER clauses in any Report Group Description within that report.

6. Within a given Report Group Description, a NEXT PAGE phrase may appear only once. If present, it must be the first LINE NUMBER clause in that Report Group Description.

7. A LINE NUMBER clause with the NEXT PAGE phrase may appear only in the description of a CONTROL HEADING, DETAIL, CONTROL FOOTING, or REPORT FOOTING report group.

8. Every entry defining a printable item must either contain a LINE NUMBER clause or be subordinate to an entry that contains a LINE NUMBER clause. See the COLUMN NUMBER clause for more information.
9. The first LINE NUMBER clause in a PAGE FOOTING report group must define an absolute line-num value.

10. line-num-plus may be zero. If line-num-plus is zero, the line will be printed on the same line as the previous print line (overprint); however, line-num-plus cannot be zero for the first print line of a report group.

**Syntax Rules (Screen Description)**

1. The LINE clause can be specified only in an elementary screen description entry.

2. identifier-1 cannot be subscripted.

**General Rules (Report Description)**

1. To establish each print line for a report group, a program must specify the LINE NUMBER clause.

2. Before presenting the print line, the Report Writer Control System (RWCS) causes line positioning as specified by a LINE NUMBER clause.

3. The NEXT PAGE phrase defines the line number of a new page on which to present the report group.

4. For a complete specification on how to determine the first print line for a report group, see Appendix D: Report Writer Presentation Rules and Tables. A partial summary of these rules follows:

   If a relative clause is not the first LINE NUMBER clause in a report group, then the line number on which its print line is presented is determined by the sum of the following:

   • The line number from the previous print line of the report group

   • line-num-plus of the relative LINE NUMBER clause

   If the first LINE NUMBER clause in the Report Group Description entry is relative and a PAGE clause is specified, the first print line for the report group is determined as follows. See the PAGE clause for the definitions of page-size, heading-line, first-detail-line, last-detail-line, and footing-line.

   a. REPORT HEADING

      The RWCS presents this group on a line number whose value is the sum of line-num of the first LINE NUMBER clause and heading-line minus 1.

   b. PAGE HEADING

      If a REPORT HEADING report group has been presented on the page on which this report group is to appear, the RWCS presents the PAGE HEADING relative to the final LINE-COUNTER setting of the REPORT HEADING.

      If no REPORT HEADING has been presented on the page, the RWCS presents this report group on the line number whose value is the sum of line-num of the first LINE NUMBER clause and heading-line minus 1.

   c. DETAIL, CONTROL HEADING, or CONTROL FOOTING

      If the value in LINE-COUNTER is less than first-detail-line, the RWCS presents the report group on first-detail-line.

      If the value in LINE-COUNTER is greater than or equal to first-detail-line and if this is the first body group to print on the page, the RWCS presents the report group on the line corresponding to the value in LINE-COUNTER.

      If the value in LINE-COUNTER is greater than or equal to first-detail-line and if this is not the first body group to print on the page, the RWCS presents the report group on the line whose value is the sum of LINE-COUNTER and line-num of the first LINE NUMBER clause of the current CONTROL HEADING, DETAIL, or CONTROL FOOTING report group.
d. PAGE FOOTING

Not applicable. The first LINE NUMBER clause of a PAGE FOOTING report group must contain an absolute line number reference.

e. REPORT FOOTING

If a PAGE FOOTING report group has been presented on the current page, the RWCS presents the REPORT FOOTING report group on the line whose value is the sum of the current value in LINE-COUNTER and line-num of the first LINE NUMBER clause of the REPORT FOOTING report group.

If no PAGE FOOTING report group has been presented on the current page, the RWCS presents the REPORT FOOTING report group on the line whose value is the sum of footing-line and line-num of the first LINE NUMBER clause of the REPORT FOOTING report group.

General Rules (Screen Description)

1. The LINE clause, in conjunction with the COLUMN clause, establishes the starting position for a screen item. This position is an offset from the starting screen coordinates specified in the ACCEPT or DISPLAY statement. The LINE clause specifies the vertical coordinate.

2. The LINE clause without the PLUS phrase specifies the absolute line number.

3. The LINE clause with the PLUS phrase specifies a line number relative to that at which the preceding item ends, regardless of whether or not the ACCEPT or DISPLAY statement displays the preceding item on the screen.

4. If the LINE clause is omitted, the following apply:
   • If no previous screen item is defined, LINE 1 is assumed.
   • If a previous screen item is defined, the ending line of that previous item is assumed.

Additional References

• COLUMN NUMBER clause
• PAGE clause
• Appendix D: Report Writer Presentation Rules and Tables
• ACCEPT statement in Chapter 6, Procedure Division
• DISPLAY statement in Chapter 6, Procedure Division

LOWLIGHT (Alpha, I64)

LOWLIGHT (Alpha, I64)

Function

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

General Format

LOWLIGHT

Syntax Rule

The LOWLIGHT clause can be specified only for an elementary screen description entry.
Additional Reference

- ACCEPT in Chapter 6, *Procedure Division*
- DISPLAY in Chapter 6, *Procedure Division*

NEXT GROUP

NEXT GROUP

Function

The NEXT GROUP clause specifies information for the vertical positioning of the next report group on a logical page following the presentation of the last line of a report group.

**General Format**

```
NEXT GROUP IS { next-group-line-num
                      PLUS next-group-line-num-plus
                      NEXT PAGE }
```

[next-group-line-num]

is a positive, 1- to 3-digit integer value greater than zero. It represents an absolute line number on a logical page and establishes a print line for the next Report Writer report group.

[next-group-line-num-plus]

is a positive, 1- to 3-digit integer value. It represents a relative line number on a logical page and establishes a print line for the next Report Writer report group.

Syntax Rules

1. A Report Group entry must not contain a NEXT GROUP clause unless the description of that Report Group contains at least one LINE NUMBER clause.

2. `next-group-line-num` and `next-group-line-num-plus` must not exceed three significant digits.

3. If a Report Description entry omits the PAGE clause, all Report Group Description entries within that report can specify a relative NEXT GROUP clause only.

4. A PAGE FOOTING Report Group must not specify the NEXT PAGE phrase of the NEXT GROUP clause.

5. A PAGE HEADING and REPORT FOOTING Report Group must not specify the NEXT GROUP clause.

General Rules

1. Page positioning occurs after the presentation of the Report Group in which the NEXT GROUP clause appears.

2. To determine a new value for LINE-COUNTER, the Report Writer Control System (RWCS) uses the vertical positioning information from the NEXT GROUP clause along with information from the TYPE and PAGE clauses, and the value in LINE-COUNTER. See Appendix D, *Report Writer Presentation Rules and Tables*.

3. The RWCS ignores the NEXT GROUP clause on a CONTROL FOOTING Report group when it detects a control break at a level other than the highest level.

4. The NEXT GROUP clause of a CONTROL HEADING, DETAIL, and CONTROL FOOTING report group refers to the next CONTROL HEADING, DETAIL, and CONTROL FOOTING to be presented, and therefore can affect the location at which the next CONTROL HEADING, DETAIL, and CONTROL FOOTING report group is presented. See Appendix D, *Report Writer Presentation Rules and Tables*. 

142
5. The NEXT GROUP clause of a REPORT HEADING report group can affect the location at which the PAGE HEADING report group is presented. See Appendix D, Report Writer Presentation Rules and Tables.

6. The NEXT GROUP clause of a PAGE FOOTING report group can affect the location at which the REPORT FOOTING report group is presented. See Appendix D, Report Writer Presentation Rules and Tables.

Additional References

- General Rules (Report Description) (General Rule 4) of the LINE NUMBER (Alpha, I64) clause
- Appendix D: Report Writer Presentation Rules and Tables

OCCURS

Function

The OCCURS clause defines tables and provides the basis for subscripting and indexing. It eliminates the need for separate entries for repeated data items.

General Format

General Format
Format 1

```
OCCURS table-size TIMES

[ { ASCENDING DESCENDING } KEY IS { key-name } ... ] ...

[ INDEXED BY { ind-name } ... ]
```

Format 2

```
OCCURS min-times TO max-times TIMES DEPENDING ON depending-item

[ { ASCENDING DESCENDING } KEY IS { key-name } ... ] ...

[ INDEXED BY { ind-name } ... ]
```

[table-size]

is an integer that specifies the exact number of occurrences of a table element.

[min-times]

is an integer that specifies the minimum number of occurrences of a table element. Its value must be greater than or equal to zero.

[max-times]

is an integer that specifies the maximum number of occurrences of a table element. Its value must be greater than min-times.

[key-name]
is the data-name of an entry that contains the OCCURS clause or an entry subordinate to it. key-name can be qualified. Each key-name after the first must name an entry subordinate to the entry that contains the OCCURS clause. The values in each key-name are the basis of the ascending or descending arrangement of the table's repeated data.

[ind-name]

is an index-name. It associates an index with the table and allows indexing in table element references.

[depending-item]

is the data-name of an elementary unsigned integer data item. Its value specifies the current number of occurrences. depending-item can be qualified.

**Syntax Rules**

1. The subject of the entry is the data-name that contains the OCCURS clause.

2. A key-name cannot contain an OCCURS clause. However, this rule does not apply to the first key-name if it is the subject of the entry.

3. There can be no OCCURS clauses between the data description entries for key-names and the subject of the entry.

4. In the OCCURS clause of the data description entry, key-name cannot be subscripted or indexed.

5. There must be an INDEXED BY phrase if any Procedure Division statements contain indexed references to the subject of the entry or to any of its subordinates.

6. The INDEXED BY phrase implicitly defines ind-name. The program cannot define ind-name elsewhere.

7. The subject of a Format 2 OCCURS clause can be followed, in the same record description, only by data description entries subordinate to it.

8. The OCCURS clause cannot be used in a data description entry that has the following:
   - A level-number of 01, 66, 77, or 88
   - A subordinate variable occurrence data item (Format 2 OCCURS clause)

9. The data item defined by depending-item cannot occupy any character position in the range delimited by the following:
   - The character position defined by the subject of the OCCURS clause
   - The last character position defined by the record description entry containing the OCCURS clause

10. Each ind-name must be a unique word in the program.

11. If the OCCURS clause is in a record description entry containing the GLOBAL clause, depending-item must refer to a global item described in the same Data Division.

12. If the OCCURS clause is in a record description entry containing the EXTERNAL clause, depending-item must refer to an external item described in the same Data Division.

**General Rules**

1. The OCCURS clause defines tables and provides the basis for subscripting and indexing.

2. Except for the OCCURS clause itself, all data description clauses associated with the subject of the OCCURS clause apply to each occurrence of the item.

3. Format 1 specifies that the subject of the entry has a fixed number of occurrences.
4. Format 2 specifies that the subject of the entry has a variable number of occurrences. min-times and max-times specify the minimum and maximum number of occurrences. Only the number of the subject's occurrences is variable; its size is fixed.

The value of depending-item must fall in the range min-times to max-times.

The contents of data items with occurrence numbers exceeding the current value of depending-item are unpredictable.

5. If a group item with a subordinate entry that has a Format 2 OCCURS clause is a sending item, the operation uses only the part of the table area specified by depending-item at the start of the operation.

If the group is a receiving item, the part of the table used is determined by the location of depending-item. If depending-item is included in the group, then the operation uses the maximum length of the group. If depending-item is not included in the group, then the operation uses only the part of the table area specified by depending-item.

6. The KEY IS phrase indicates that the repeated data is arranged in ascending or descending order according to the values in the data items named by key-name. The rules for operand comparison determine the ascending or descending order. The position of each key-name in the list determines its significance. The first is the most significant, and the last is least significant.

7. If a Format 2 OCCURS clause is in a record description entry and the associated file description entry has the VARYING phrase of the RECORD clause, the records are variable length.

If the RECORD clause does not have the DEPENDING ON phrase, the program must set the OCCURS clause depending-item to the number of occurrences before executing a RELEASE, REWRITE, or WRITE statement. The depending-item value determines the length of the record to be written.

**Technical Note**

If the subject of the OCCURS clause (or any of its subordinates) has the SYNCHRONIZED clause, the length of the subject of the OCCURS clause, or the group containing it, could increase. SYNCHRONIZED clause alignment can add fill bytes to the group containing the subject of the OCCURS clause and to the subject itself.

**Additional References**

- SEARCH statement in Chapter 6, *Procedure Division*
- Section 5.2.3: Additional Alignment Rules for Record Allocation
- Section 6.5.1.1: Comparison of Numeric Operands section in Chapter 6, *Procedure Division*
- Section 6.5.1.2: Comparison of Nonnumeric Operands section in Chapter 6, *Procedure Division*

**Examples**

1. One-dimensional table:

   This record description entry describes a 20-character record. The record contains 10 occurrences of ITEMB, a 2-character data item.

   01 ITEMA.
   03 ITEMB OCCURS 10 TIMES PIC XX.

2. Two-dimensional table:

   This record description entry describes a 320-character record. The record contains 8 occurrences of ITEMB, a 40-character data item. ITEMB contains 10 occurrences of ITEMC, a 4-character data item. Each ITEMC contains 2 data items: ITEMD and ITEME.

   01 ITEMA.
03 ITEMB OCCURS 8 TIMES.
   05 ITEMC OCCURS 10 TIMES.
   07 ITEMD PIC X.
   07 ITEME PIC XXX.

ITEMB (1) refers to a 40-character data item, the first 10 occurrences of ITEMC. Similarly, ITEMB (5) refers to the fifth group of 10 occurrences of ITEMC.

ITEME (3,4) refers to ITEME in the fourth occurrence of ITEMC in the third occurrence of ITEMB:

3. Variable occurrence data item:

When ITEMA is a receiving item, its size is 2128 characters. When it is a sending item, its size can vary from 70 to 2128 characters, depending on the value in ITEMC.

Each ITEME is 42 characters long. Its size cannot change. The only effect of the value of ITEMC is to determine the number of ITEME occurrences.

There are 10 occurrences of ITEMH and ITEMI in each occurrence of ITEME.

01 ITEMA.
   03 ITEMB PIC X(6).
   03 ITEMC PIC 99.
   03 ITEMD PIC X(20).
   03 ITEME OCCURS 1 TO 50 TIMES DEPENDING ON ITEMC.
      05 ITEMF PIC XX.
      05 ITEMG OCCURS 10 TIMES.
         07 ITEMH PIC X.
         07 ITEMI PIC XXX.

PAGE

Function

The PAGE clause defines the length of a logical page and the vertical subdivisions within which report groups are presented.

General Format

```
PAGE page-size
   [ LIMITS ARE line-size ]
   [ LISTING heading-line ]
   [ FIRST DETAIL fist-detail-line ]
   [ LAST DETAIL last-detail-line ]
   [ FOOTING footing-line ]
```

[page-size]

is a 1- to 3-digit integer. It defines the number of lines available on a logical page.
[heading-line]

is a 1- to 3-digit integer. It defines the first line number for a REPORT HEADING or PAGE HEADING report group on the logical page.

[first-detail-line]

is a 1- to 3-digit integer. It defines the first line number for a CONTROL HEADING, DETAIL, and CONTROL FOOTING report group on the logical page.

[last-detail-line]

is a 1- to 3-digit integer. It defines the last line number for a CONTROL HEADING or DETAIL report group on the logical page.

[footing-line]

is a 1- to 3-digit integer. It defines the last line number for a CONTROL FOOTING report group and the first line number for the PAGE FOOTING report group on the logical page.

Syntax Rules

1. The HEADING, FIRST DETAIL, LAST DETAIL, and FOOTING phrases may be written in any order.

2. page-size must not exceed three significant digits and must be greater than or equal to footing-line.

3. heading-line must be greater than or equal to one.

4. first-detail-line must be greater than or equal to heading-line.

5. last-detail-line must be greater than or equal to first-detail-line.

6. footing-line must be greater than or equal to last-detail-line.

7. The rules in Table 5.6, “Page Regions Established by the PAGE Clause” summarize the rules presented in Appendix D: Report Writer Presentation Rules and Tables. They indicate the vertical subdivision of the page in which each type of report group may appear when the PAGE clause is specified.

   a. To present a REPORT HEADING report group on a page by itself (NEXT GROUP NEXT PAGE), define the REPORT HEADING clause to be in the vertical subdivision of the page extending from heading-line to page-size, inclusive.

   To present a REPORT HEADING report group on a page with other report groups, define the REPORT HEADING clause to be in the vertical subdivision of the page extending from heading-line to first-detail-line minus one, inclusive.

   b. A PAGE HEADING clause must be defined in the vertical subdivision of the page extending from heading-line to first-detail-line minus one, inclusive.

   c. A CONTROL HEADING or DETAIL clause must be defined in the vertical subdivision of the page extending from first-detail-line to last-detail-line, inclusive.

   d. A CONTROL FOOTING clause must be defined in the vertical subdivision of the page extending from first-detail-line to footing-line, inclusive.

   e. A PAGE FOOTING clause must be defined in the vertical subdivision of the page extending from footing-line plus one to page-size, inclusive.

   f. To present a REPORT FOOTING report group on a page by itself, define the REPORT FOOTING clause in the vertical subdivision of the page extending from heading-line to page-size, inclusive.

   To present a REPORT FOOTING report group on a page with other report groups, define the REPORT FOOTING clause in the vertical subdivision of the page extending from footing-line plus one to page-size.
8. All report groups must be defined such that they can be presented on one logical page. The Report Writer Control System (RWCS) never splits a multiline report group across logical page boundaries.

**General Rules**

1. REPORT HEADING and PAGE HEADING report groups may not be presented on or beyond the first-detail-line.

2. PAGE FOOTING and REPORT FOOTING report groups must follow the footing-line.

3. If the PAGE clause is specified, the following implicit default values are assumed for any omitted phrases:
   
   a. If the HEADING phrase is omitted, heading-line equals one.
   
   b. If the FIRST DETAIL phrase is omitted, first-detail-line equals heading-line.
   
   c. If the LAST DETAIL and FOOTING phrases are both omitted, last-detail-line and footing-line equal page-size.
   
   d. If the FOOTING phrase is specified and the LAST DETAIL phrase is omitted, last-detail-line equals footing-line.
   
   e. If the LAST DETAIL phrase is specified and the FOOTING phrase is omitted, footing-line equals last-detail-line.

4. If the PAGE clause is omitted, the report consists of a single page of infinite length with relative line numbering.

5. If a REPORT HEADER report group is to appear on a page with other report groups, the first line following the heading report groups (REPORT HEADER and PAGE HEADER) must be blank.

6. If a REPORT FOOTING report group is to appear on a page with other report groups, the first line preceding the footing report groups (PAGE FOOTING and REPORT FOOTING) must be blank.

**Additional References**

- General Rules (Report Description) (General Rule 4) of the LINE NUMBER (Alpha, I64) clause
- Appendix D: Report Writer Presentation Rules and Tables

Table 5.6, “Page Regions Established by the PAGE Clause” shows the page regions established by the PAGE clause.

**Table 5.6. Page Regions Established by the PAGE Clause**

<table>
<thead>
<tr>
<th>Region Boundaries</th>
<th>Line Positioning for the First Report Group Within the Region</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First Line Number of the Region</strong></td>
<td><strong>Last Line Number of the Region</strong></td>
</tr>
<tr>
<td>Report Heading Described with NEXT GROUP NEXT PAGE</td>
<td>heading-line</td>
</tr>
<tr>
<td>Report Footing Described with LINE line-num NEXT PAGE</td>
<td>heading-line</td>
</tr>
<tr>
<td>Page Heading</td>
<td>heading-line</td>
</tr>
<tr>
<td>Report Heading Not Described with NEXT GROUP NEXT PAGE</td>
<td>heading-line</td>
</tr>
</tbody>
</table>
Data Division

Region Boundaries

<table>
<thead>
<tr>
<th>Report Groups that Can Be Presented in a Region</th>
<th>First Line Number of the Region</th>
<th>Last Line Number of the Region</th>
<th>Line Positioning for the First Report Group Within the Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Heading</td>
<td>first-detail-line</td>
<td>last-detail-line</td>
<td>If LINE-COUNTER is greater than or equal to first-detail-line, position on LINE-COUNTER plus 1</td>
</tr>
<tr>
<td>Detail</td>
<td></td>
<td></td>
<td>If LINE-COUNTER is less than first-detail-line, position on first-detail-line</td>
</tr>
<tr>
<td>Control Footing</td>
<td>first-detail-line</td>
<td>footing-line</td>
<td>Same as preceding</td>
</tr>
<tr>
<td>Page Footing</td>
<td>footing line</td>
<td>page-size</td>
<td></td>
</tr>
<tr>
<td>Report Footing Not Described with LINE line-num NEXT PAGE</td>
<td>plus 1</td>
<td></td>
<td>LINE-NUMBER plus footing-line</td>
</tr>
</tbody>
</table>

PICTURE

Function

The PICTURE clause specifies the general characteristics and editing requirements of an elementary item, including an elementary screen item.

General Formats

Format 1

\[
\{\text{PICTURE} \\
\text{PIC} \} \text{ IS character-string}
\]

Format 2 (Screen Section)

\[
\{\text{PICTURE} \\
\text{PIC} \} \text{ IS character-string} \{\text{USING identifier-3} \\
\{\text{FROM} \{\text{identifier-4} \{\text{REPEAT-1} \} \} \\
\text{TO} \{\text{identifier-5} \} \} \}
\]

Syntax Rules (Both Formats)

1. You can use the PICTURE clause only for an elementary item.

2. character-string contains allowable combinations of characters in the COBOL character set. These characters are called the symbols of the PICTURE character-string.

3. character-string can contain from 1 to 255 symbols.

4. PIC is an abbreviation for PICTURE.

5. The asterisk (*), when used as a zero suppression symbol, and the BLANK WHEN ZERO clause cannot be used in the same entry.
Syntax Rule (Format 1)

1. The PICTURE clause is required for every elementary item except an item specified by the USAGE IS BINARY-CHAR, BINARY-SHORT, BINARY-LONG, BINARY-DOUBLE, COMP-1, COMP-2, FLOAT-SHORT, FLOAT-LONG, FLOAT-EXTENDED, POINTER, or INDEX clause and the subject of a RENAMES clause. Data description entries for these items cannot contain a PICTURE clause.

Syntax Rules (Format 2)

1. The PICTURE clause for a numeric screen item must either define a numeric edited item or must contain only “9”s and an optional “S”.

2. Each PICTURE clause in a screen description entry must contain a FROM or a TO phrase, or both, or a USING phrase.

3. In a screen description entry, if the PICTURE clause is specified, the VALUE clause cannot be specified.

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

General Rules (Both Formats)

1. The PICTURE clause categorizes a data item or screen item and determines what the item can contain. In the case of a PICTURE clause containing all Xs, the USAGE clause determines whether the item is alphanumeric or numeric. Table 5.7, “Summary of PICTURE Clause Rules” shows the valid contents of both character-string and the item itself for each category. The general rules following this table supplement the information it contains.

Table 5.7. Summary of PICTURE Clause Rules

<table>
<thead>
<tr>
<th>Category of Receiving Item</th>
<th>PICTURE of Receiving Item</th>
<th>Valid Contents of Sending Item</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabetic</td>
<td>Must contain one or more As.</td>
<td>One or more alphabetic characters.</td>
<td>AA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A (9)</td>
</tr>
<tr>
<td>Numeric</td>
<td>Must contain at least one 9. May contain P's, one S, and one V. If USAGE IS COMP-5 or USAGE IS COMP-X, may contain all Xs.</td>
<td>One or more numeric characters.</td>
<td>S9(4)V99</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9PPP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SPP9</td>
</tr>
<tr>
<td>Alphanumeric</td>
<td>Must contain combinations of As, Xs, and 9s. Can be all Xs. Cannot be all As or all 9s.</td>
<td>One or more characters in computer character set.</td>
<td>XX99XX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AAXA(4)</td>
</tr>
<tr>
<td>Alphanumeric Edited</td>
<td>Must contain at least one A or X. Must also contain at least one B, 0, or /. Can contain one or more 9s.</td>
<td>One or more characters in computer character set.</td>
<td>XXBXXB9(4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>XX/99/00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9(6)/X</td>
</tr>
<tr>
<td>Numeric Edited</td>
<td>Must contain at least one 0, B, /, Z, *, +, (comma), ., −, CR, DB, or cs. Can contain Ps, 9s, and one V. Must describe 1 to 31 digit positions, which can be represented by 9s, zero suppression symbols (Z,</td>
<td>One or more numeric characters.</td>
<td><em>,</em><strong>,</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ZZ,ZZZ/9(4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$S,$$$DB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$9,999CR</td>
</tr>
</tbody>
</table>
Note

COMP-1 and COMP-2 data items are numeric. However, their data description entries cannot have a PICTURE clause.

2. In an alphanumeric item definition, each character position is treated as if it were represented by an X, even though A or 9 may be specified.

3. Some PICTURE symbols represent character positions and some do not. An item's size is determined by adding up all the symbols that represent a character position. For example, a numeric item with a PICTURE of 999V99 has a size of five characters. The symbol V does not count toward the item's size.

4. character-string can contain a repeat count to represent consecutive occurrences of the following symbols: A, the comma (,), X, 9, P, Z, *, B, /, 0, +, –, and the currency symbol (cs). The repeat count must be an unsigned, nonzero integer enclosed in parentheses. For example, S9(6)V9(4) is equivalent to S999999V9999. However, character-string can contain no more than one of the following symbols: S, V, a period (.), CR, and DB.

5. The PICTURE clause symbols and their functions appear in Table 5.8, “PICTURE Clause Symbols”.

### Table 5.8. PICTURE Clause Symbols

<table>
<thead>
<tr>
<th>Picture Clause Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Represents a character position that can contain only an alphabetic character. An alphabetic character belongs to the set of characters: A to Z, a to z, and the space. Can occur more than once. Counts toward the size of the item.</td>
</tr>
<tr>
<td>B</td>
<td>Represents a character position into which a space is inserted. Can occur more than once. Counts toward the size of the item.</td>
</tr>
<tr>
<td>N</td>
<td>For USAGE IS DISPLAY, represents a character position that can contain any 2-byte character from the national character set. This is available only if/NATIONALITY=JAPAN or -nationality japan is specified.</td>
</tr>
<tr>
<td>P</td>
<td>Specifies an assumed decimal scaling position, defining the location of the decimal point when one is not specified in character-string. Can occur more than once, but only as a contiguous string of Ps at either the leftmost or rightmost end (not both) of character-string. The assumed decimal point character (V) is redundant when specified. However, when it is specified, V can appear to the left of the leftmost P or to the right of the rightmost P. Does not count toward the size of the item. However, each P counts toward the maximum number of digit positions (31) in a numeric or numeric edited item. Cannot be used if an explicit decimal point (.) appears in character-string.</td>
</tr>
</tbody>
</table>
In certain operations that refer to an item with P characters in *character-string*, the compiler treats each P position as if it contained the value zero. For example, an item with PICTURE 99PPP can have 100 unique values that range from 0 to 99,000 (0, 1000, 2000, …, 99,000). An item with PICTURE PP9 can have 10 unique values (0, .001, .002, …, .009). These operations are any of the following:

- Any operation requiring a numeric sending operand
- A MOVE statement where the sending operand is numeric and its PICTURE *character-string* contains the symbol P
- A MOVE statement where the sending operand is numeric edited and its PICTURE *character-string* contains the symbol P, and the receiving operand is numeric or numeric edited
- A comparison operation where both operands are numeric

In all other operations, the compiler ignores the digit positions specified with the symbol P and does not count them toward the size of the operand.

<table>
<thead>
<tr>
<th>Picture Clause Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S</strong></td>
<td>Indicates the presence of an operational sign, but does not specify the sign representation or position. Can occur only once, as the leftmost character in <em>character-string</em>. Does not count toward the size of the item unless the data or screen description entry contains a SIGN IS SEPARATE clause. If the SIGN clause does not appear in the item's data description, S is equivalent to SIGN IS TRAILING.</td>
</tr>
<tr>
<td><strong>V</strong></td>
<td>Specifies the location of the assumed decimal point. Can occur only once. Does not count toward the size of the item. Cannot be used if an explicit decimal point (.) appears in the PICTURE.</td>
</tr>
<tr>
<td><strong>X</strong></td>
<td>For USAGE IS DISPLAY, represents a character position that can contain any character from the computer character set. For USAGE IS COMP-5 or USAGE IS COMP-X, represents a byte of computer storage. Can occur more than once. Counts toward the size of the item.</td>
</tr>
<tr>
<td><strong>Z</strong></td>
<td>Represents a leading digit position that is replaced by a space when its value and the value of the digits to its left are zero. Can occur more than once. Counts toward the size of the item. Use of Z excludes the use of the asterisk (*) for zero suppression and replacement.</td>
</tr>
<tr>
<td><strong>9</strong></td>
<td>Represents a digit position that can contain only the digits 0 to 9. Can occur more than once. Counts toward the size of the item.</td>
</tr>
<tr>
<td><strong>0</strong></td>
<td>Represents a character position into which 0 is inserted. Can occur more than once. Counts toward the size of the item.</td>
</tr>
</tbody>
</table>
**Data Division**

<table>
<thead>
<tr>
<th>Picture Clause Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Represents a character position into which a slash (/) is inserted.</td>
</tr>
<tr>
<td></td>
<td>Can occur more than once.</td>
</tr>
<tr>
<td></td>
<td>Counts toward the size of the item.</td>
</tr>
<tr>
<td>,</td>
<td>Represents a character position into which a comma (,) is inserted.</td>
</tr>
<tr>
<td></td>
<td>Can occur more than once.</td>
</tr>
<tr>
<td></td>
<td>Counts toward the size of the item.</td>
</tr>
<tr>
<td>.</td>
<td>Represents a character position into which a decimal point (.) is inserted.</td>
</tr>
<tr>
<td></td>
<td>It also represents the decimal point for alignment purposes.</td>
</tr>
<tr>
<td></td>
<td>Can occur only once.</td>
</tr>
<tr>
<td></td>
<td>Counts toward the size of the item.</td>
</tr>
<tr>
<td>+ –</td>
<td>Represents the editing sign control symbols, the plus sign (+) and minus sign (-).</td>
</tr>
<tr>
<td></td>
<td>Each can occur more than once.</td>
</tr>
<tr>
<td></td>
<td>Each counts as one character toward the size of the item.</td>
</tr>
<tr>
<td></td>
<td>character-string can contain either a plus sign (+) or minus (-), but not both. Also, the use of either character excludes the use of both CR and DB.</td>
</tr>
<tr>
<td>CR DB</td>
<td>Represents the editing sign control symbols, credit (CR) and debit (DB).</td>
</tr>
<tr>
<td></td>
<td>Each can occur only once, as the two rightmost character positions.</td>
</tr>
<tr>
<td></td>
<td>Each counts as two characters toward the size of the item.</td>
</tr>
<tr>
<td></td>
<td>character-string can contain either CR or DB, but not both. Also, the use of either excludes the use of the plus sign (+) and minus sign (-) as fixed insertion characters.</td>
</tr>
<tr>
<td>*</td>
<td>Represents a leading digit position that is replaced by an asterisk (*) when its value and the values of all digit positions to its left are zero.</td>
</tr>
<tr>
<td></td>
<td>Can occur more than once.</td>
</tr>
<tr>
<td></td>
<td>Counts toward the size of the item.</td>
</tr>
<tr>
<td></td>
<td>Use of an asterisk (*) excludes the use of Z for zero suppression and replacement.</td>
</tr>
<tr>
<td>cs</td>
<td>Represents a character position into which the currency symbol is inserted. This symbol is either the currency sign ($) or the character specified in the CURRENCY SIGN clause of the SPECIAL-NAMES paragraph or (on OpenVMS) the character specified at DCL command level in the definition of the SYSSCURRENCY logical name.</td>
</tr>
<tr>
<td></td>
<td>Can occur more than once.</td>
</tr>
<tr>
<td></td>
<td>Counts as one character toward the size of the item.</td>
</tr>
</tbody>
</table>

---

**General Rules (Format 2)**

1. The USING, FROM, and TO phrases have meaning only when the screen item’s name or a screen name in its hierarchy, is specified in an ACCEPT or DISPLAY statement.

2. 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 TO phrases must be used in the PICTURE clause of the screen item.

---

**Note:** When a program contains the DECIMAL POINT IS COMMA clause, the functions and rules for the period (.) and comma (,) are exchanged. In other words, the rules that apply to the period apply to the comma, and vice versa.
3. 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 item.

4. identifier-3, identifier-4, identifier-5, and literal-1 need not be the same length as the screen item containing the PICTURE clause.

5. Transfers between identifier-3, identifier-4, identifier-5, and literal-1, on the one hand, and the screen item are made in accordance with the rules of the MOVE statement. (See the MOVE Statement in Chapter 6, Procedure Division.)

6. When the FROM phrase is specified:
   a. On DISPLAY statement execution, data is transferred from identifier-4 or literal-1, after being edited in accordance with character-string, 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 starting screen coordinates specified in the DISPLAY statement.
   b. The FROM phrase has no meaning in the execution of an ACCEPT statement.

7. When the TO phrase is specified:
   a. At ACCEPT statement completion, the data entered into the field on the screen is transferred to identifier-5, after being edited in accordance with the picture string specified for identifier-5.
   b. The TO phrase has no meaning in the execution of a DISPLAY statement.

8. When the USING phrase, or the FROM and TO phrases are specified:
   a. On DISPLAY statement execution, data is transferred from identifier-3, identifier-4, or literal-1 as described in rule 11a above.
   b. On ACCEPT statement execution, data is transferred from identifier-3, identifier-4, or literal-1 as described in rule 11a above. At ACCEPT statement completion, the data entered into the screen item is transferred to identifier-3 or identifier-5 as described in rule 12a above.

### Editing Rules

1. There are two PICTURE clause editing methods: insertion editing and suppression and replacement editing. Each method has the following variations:

<table>
<thead>
<tr>
<th>Editing Method</th>
<th>Variations in Each Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion</td>
<td>Simple insertion editing, special insertion editing, fixed insertion editing, or floating insertion editing</td>
</tr>
<tr>
<td>Suppression and Replacement</td>
<td>Zero suppression and replacement with spaces, or zero suppression and replacement with asterisks</td>
</tr>
</tbody>
</table>

2. The types of editing that a program can perform on an item depend on the item's category:

<table>
<thead>
<tr>
<th>Category</th>
<th>Types of Editing</th>
<th>Valid Editing Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabetic</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Numeric</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Alphanumeric</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Alphanumeric Edited</td>
<td>Simple insertion</td>
<td>0, B, and /</td>
</tr>
<tr>
<td>Numeric Edited</td>
<td>All</td>
<td>All, subject to Editing Rule 3</td>
</tr>
</tbody>
</table>

3. Floating insertion editing and editing by zero suppression and replacement are mutually exclusive. That is, a PICTURE clause can use one type of editing or the other, but not both.
Furthermore, a PICTURE clause can use only one type of replacement symbol for zero suppression. The space \( (Z) \) and asterisk \( (*) \) symbols cannot appear in the same PICTURE clause.

### Simple Insertion Editing

1. A comma (,) space (B), zero (0), and slash (/) are symbols you can use in simple insertion editing. They indicate an item position to contain the character they represent. These symbols count toward the size of the item.

   If the comma is the last symbol in character-string, the PICTURE clause must be the last clause of the data description entry. In this case, a comma followed by a period (,.) are the last two characters of the data description entry. However, if the DECIMAL-POINT IS COMMA clause is in the SPECIAL-NAMES paragraph, the data description entry ends with two consecutive periods.

### Special Insertion Editing

1. The period (.) is the only symbol used in special insertion editing. It represents the item position to contain the actual decimal point; however, it also represents the decimal point for alignment purposes. Therefore, the assumed decimal point (V) and the actual decimal point (.) cannot be used in the same character-string. The period counts toward the size of the item.

   If the period is the last symbol in character-string, the PICTURE clause must be the last clause of the data description entry. In this case, the data description entry ends with two periods. However, if the DECIMAL-POINT IS COMMA clause is in the SPECIAL-NAMES paragraph, a comma followed by period (,.) are the last two characters of the data description entry.

### Fixed Insertion Editing

1. The currency symbol (cs) and the editing sign control symbols (+, –, CR, and DB) are the symbols used in fixed insertion editing. character-string can contain only one currency symbol and only one of the editing sign control symbols as fixed insertion characters.

   CR and DB each represent two character positions, which must be the two rightmost positions.

   The plus sign (+) and minus sign (-) must be either the leftmost or rightmost character position that counts toward the size of the item.

   The currency symbol (cs) must be the leftmost character position that counts toward the size of the item; however, a plus sign (+) or minus sign (-) can precede it.

   Fixed insertion editing causes the insertion symbol to occupy the same position in the edited item as in character-string. Table 5.9, “Using Sign Control Symbols in Fixed Insertion Editing” shows that the results of using editing sign control symbols depend on the item’s value.

### Table 5.9. Using Sign Control Symbols in Fixed Insertion Editing

<table>
<thead>
<tr>
<th>Editing Symbol in PICTURE</th>
<th>Item Positive or Zero</th>
<th>Item Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>–</td>
</tr>
<tr>
<td>–</td>
<td>space</td>
<td>–</td>
</tr>
<tr>
<td>CR</td>
<td>2 spaces</td>
<td>CR</td>
</tr>
<tr>
<td>DB</td>
<td>2 spaces</td>
<td>DB</td>
</tr>
</tbody>
</table>

### Floating Insertion Editing
1. The currency symbol (cs), the plus sign (+), and the minus sign (-) are the symbols used in floating insertion editing. They are mutually exclusive in character-string. That is, if any floating insertion symbol appears in character-string, no other floating insertion symbol can appear.

To indicate floating insertion editing, you must use a string of at least two floating insertion symbols. You can include simple insertion symbols either within the floating string or immediately to the right of the floating string. These simple insertion symbols are treated as part of the floating string. That is, they appear in results only when the value of the item is large enough to include the position occupied by the simple insertion symbol. You can append the fixed insertion symbols CR or DB immediately to the right of a floating string.

The leftmost symbol of the floating insertion string represents the leftmost position in which a floating insertion character can appear. This character position cannot be filled by a digit.

The second floating symbol from the left represents the leftmost limit of the numeric data the item can store. Nonzero numeric characters can replace all symbols at or to the right of this limit.

You can use the floating insertion symbol in only two ways. It can represent the following:

a. Any or all leading numeric character positions to the left of the decimal point

In this case, run-time results show a single insertion character in the position immediately preceding either the first nonzero digit in the item or the decimal point, whichever appears leftmost in the data. For example, an item whose PICTURE is $$$.99 and whose value is zero appears as $.00.

b. All numeric character positions in the PICTURE character-string

In this case, you must specify at least one insertion symbol to the left of the decimal point. When the item has a nonzero value, run-time results are the same as when all the insertion symbols are to the left of the decimal point. However, when the item has a zero value, run-time results show neither a floating insertion character nor the decimal point. For example, an item whose PICTURE is $$$.$$ and whose value is zero appears as spaces.

If the floating insertion symbol is a plus sign (+) or minus sign (-), the actual character inserted depends on the value of the item. Table 5.10, “Using Sign Control Symbols in Floating Insertion Editing” shows the possible results of using editing sign control symbols in floating insertion editing.

<table>
<thead>
<tr>
<th>Editing Symbol in PICTURE Character-String</th>
<th>Item Positive or Zero</th>
<th>Item Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>–</td>
</tr>
<tr>
<td>–</td>
<td>space</td>
<td>–</td>
</tr>
</tbody>
</table>

To avoid truncation, the minimum size of character-string must be the sum of:

- The number of characters in the sending item
- The number of simple, special, or fixed insertion characters edited into the receiving item
- One, for the floating insertion character

### Zero Suppression and Replacement Editing

1. One or more occurrences of the space symbol (Z) or the asterisk (*) define a floating suppression string, which can suppress leading zeros in numeric character positions. The space symbol (Z) causes spaces to replace the zeros; an asterisk (*) causes asterisks to replace the zeros.
The suppression symbols are mutually exclusive. That is, *character-string* can contain either the space symbol (Z) or the asterisk (*), but not both.

Each suppression symbol counts toward the size of the item.

You can include simple insertion symbols either within the floating string or immediately to its right. These simple insertion symbols are treated as part of the floating string. That is, they appear in results only when the value of the item is large enough to include a position occupied by a simple insertion symbol.

You can use zero suppression symbols to represent either:

- Any or all leading numeric character positions to the left of the decimal point
- All numeric character positions on both sides of the decimal point

For example, both ZZZ9.99 and ZZ.ZZ are valid *character-strings*, but ZZZ.Z9 is not.

The following actions occur if the suppression symbols represent any or all leading numeric character positions to the left of the decimal point:

- The replacement character replaces any leading zero in the data that corresponds to a suppression symbol in the string.
- Suppression ends at either the first nonzero digit in the data represented by the suppression string or at the decimal point, whichever appears first in the data.

The following events occur if the suppression symbols represent all numeric positions in *character-string*:

- If the value of the data is not zero, the result is the same as if all suppression symbols were to the left of the decimal point. That is, zeros to the right of the decimal point are not suppressed.
- If the value is zero and the suppression symbol is a Z, all character positions in the edited item (including any editing characters) contain spaces.
- If the value is zero and the suppression symbol is an asterisk (*), all character positions in the edited item (including any insertion editing characters other than the decimal point) contain asterisks. The decimal point appears in the item.

2. The plus sign (+), minus sign (-), asterisk (*), space (Z), and currency symbol (cs) are mutually exclusive when they are used as floating replacement characters. That is, if any one of these symbols appears as a floating replacement character, none of the other symbols can appear as a floating replacement character in the same PICTURE clause.

**PICTURE Symbol Precedence Rules**

1. *character-string* must contain either:

   - At least one of the symbols A, X, Z, 9, or asterisk (*)
   - At least two of the symbols plus sign (+), minus sign (-), or currency symbol (cs)

2. Figure 5.10, “PICTURE Symbol Precedence Rules” summarizes the rules for combining symbols to form *character-strings* more complex than the basic possibilities listed in rule 1. The table shows that the use of one symbol in a *character-string* excludes the use of certain others before or after it.

The table uses the following conventions:

- A Y at an intersection means the symbols at the top of the column (*First Symbol*) can precede the symbols at the left of the row (*Second Symbol*).
• Braces ({ }) enclose symbols that are mutually exclusive.

• The currency symbol appears as cs.

• Symbols appear twice in a column or row when their rules of use depend upon their location in a character-string. These double entry symbols are as follows:
  
  • Fixed insertion symbols (+ and –)
  
  • Floating symbols Z, asterisk (*), plus sign (+), minus sign (‐), and currency symbol (cs)

  • P

The uppermost entry in a column (or the leftmost entry in a row) represents symbol use left of the actual or implied decimal point position. The second entry represents symbol use to the right of the decimal point.

**Figure 5.10. PICTURE Symbol Precedence Rules**

<table>
<thead>
<tr>
<th>Second Symbol</th>
<th>Nonfloating Insertion Symbols</th>
<th>Floating Insertion Symbols</th>
<th>Other Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td>O</td>
<td>‐</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-floating Insertion Symbols</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Floating Insertion Symbols</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Other Symbols</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

**Addition References**

• SIGN clause

• MOVE statement in Chapter 6, Procedure Division

• ACCEPT statement in Chapter 6, Procedure Division

• DISPLAY statement in Chapter 6, Procedure Division

**Examples**

The Procedure Division entry for the MOVE statement contains examples that illustrate this clause.

**RECORD**

RECORD

**Function**
The RECORD clause specifies the number of character positions in either a fixed- or variable-length record. If the number of character positions does not vary, the RECORD clause specifies the minimum and maximum number of character positions in a variable-length record.

**General Format**

**Format 1**

```
RECORD CONTAINS [ shortest-rec TO ] longest-rec CHARACTERS
```

**Format 2**

```
RECORD IS VARYING IN SIZE

[ FROM shortest-rec ] [ TO longest-rec ] CHARACTERS

DEPENDING ON depending-item ]
```

[shortest-rec] is an integer that specifies the minimum number of character positions in a variable-length record. Its value must be greater than or equal to zero.

[longest-rec] is an integer greater than shortest-rec. It specifies the maximum number of character positions in a variable-length record or the size of a fixed-length record.

[depending-item] is the data-name of an elementary unsigned integer data item in the Working-Storage or Linkage Section. It specifies the number of character positions for an output operation, and it contains the number of character positions after a successful input operation.

**Syntax Rules**

1. No record description entry for a file can specify the following:
   - Fewer character positions than shortest-rec
   - More character positions than longest-rec

2. In a sort-merge file description entry, the first shortest-rec character positions of the record must be large enough to include all keys specified in any SORT or MERGE statement for the sort or merge file.

3. For an indexed file, the first shortest-rec character positions of the record must be large enough to include all record keys.

4. If the DEPENDING ON phrase is present and if the associated file connector is an external file connector, depending-item must have the external attribute and must specify the same data-name in all file description entries associated with the external file connector.

**General Rules**

**Both Formats**

1. The absence of a RECORD clause is the same as a Format 1 RECORD clause with no shortest-rec phrase and with longest-rec equal to the greatest number of character positions described for any of the file's records.

2. The number of characters described by a record description entry is the sum of both of the following:
   - The number of character positions in all elementary items excluding redefinitions and renamings
• The number of fill bytes added because of alignment requirements

If the record description entry contains a table definition, the sum includes the number of character positions in the maximum number of table elements.

3. If the associated file connector is an external file connector, all file description entries in the run unit associated with that file connector must define the same values for shortest-rec and longest-rec. If the RECORD clause is not specified, all record description entries associated with this file connector must be the same length.

**Format 1**

1. If there is no shortest-rec phrase, Format 1 specifies fixed-length records. longest-rec then specifies the number of character positions in each record of the file.

2. If there is a shortest-rec phrase, Format 1 specifies variable-length records, the same as Format 2 without the DEPENDING phrase.

3. For variable-length records:
   • The maximum record size for a READ or RETURN operation is the number of character positions described in the largest record description entry for the file.
   • During execution of a RELEASE, REWRITE, or WRITE statement, the number of character positions in a record equals the number of character positions in the record description entry referred to by the statement.
   • If all record description entries for the file describe records of the same size, RELEASE, REWRITE, and WRITE statements for the file transfer fixed-length records in variable-length format.

**Format 2**

1. Format 2 specifies variable-length records.

2. If the clause does not contain shortest-rec, the minimum number of character positions in any of the file's records is the least number of character positions described by a record description entry for the file.

3. If the clause does not contain longest-rec, the maximum number of character positions in any of the file's records is the greatest number of character positions described by a record description entry for the file.

4. If there is a DEPENDING phrase, the program must set depending-item to the number of character positions in the record before executing a RELEASE, REWRITE, or WRITE statement for the file.

5. After successful execution of a READ or RETURN statement for the file, the value of depending-item indicates the number of character positions in the accessed record.

6. The depending-item value is not changed by executions of:
   • DELETE and START statements
   • Unsuccessful READ and RETURN statements

7. For RELEASE, REWRITE, and WRITE statement execution, determining the number of character positions in the record depends partly upon whether or not the record contains a variable occurrence item (an item described by the OCCURS clause or one that is subordinate to another item so described). During execution of these statements, three rules determine the number of character positions in the record:
   • If there is a depending-item, its value specifies the number of character positions.
   • If there is no depending-item and the record does not contain a variable occurrence item, the number of character positions described by the record description entry specifies the number of character positions.
• If there is no depending-item and the record contains a variable occurrence item, the number of character positions is the sum of the character positions in the fixed part of the record and the table elements specified by the OCCURS clause depending-item when the output statement executes.

### Additional References

- EXTERNAL clause
- SYNCHRONIZED clause
- USAGE clause
- Data Description

### RECORD KEY

#### RECORD KEY

#### Function

The RECORD KEY clause specifies the Prime Record Key access path to indexed file records.

**General Format**

```
RECORD KEY IS { rec-key | WITH DUPLICATES | [ ASCENDING | DESCENDING ] } [ seg-key ]
```

[rec-key]

is the Record Key for the file. It is the data-name of a data item in a record description entry for the file. It can be qualified, but it cannot be a group item that contains a variable-occurrence data item. The data item must be described as one of the following:

- Alphanumeric item
- Alphabetic item
- Group item
- Unsigned numeric display item
- COMP-3 integer
- COMP integer

[seg-key]

is a segmented-key name that represents the concatenation of one or more (up to eight) occurrences of seg.

[seg]

is the data-name of a data item in a record description entry for the file. It can be qualified, but it cannot be a group item that contains a variable-occurrence data item. The data item must be described as one of the following:

- Alphanumeric item
- Alphabetic item
- Group item
• Unsigned numeric display item

**Syntax Rule**

The RECORD KEY clause is required for indexed files. It can be in either the file description entry or in the file's Environment Division SELECT clause. However, it cannot be in both the SELECT clause and the file description entry for the same file.

**General Rules**

1. `seg-key` is the concatenation of all specified key segments in the order specified.

2. `seg-key` can be referenced only in a READ (Format 3) or START statement.

3. The RECORD KEY clause specifies the Prime Record Key for a file.

4. The order of keys, whether ASCENDING or DESCENDING, must be the same as the order used when the file was created.

5. Each key can be specified as ASCENDING or DESCENDING (ASCENDING is the default). In an ASCENDING key, lower key values occur toward the beginning of the sorted file. In a DESCENDING key, higher key values occur toward the beginning of the sorted file.

6. The data description of `rec-key`, or the segments of `seg-key`, and their relative locations in the record, must be the same as those used when the file was created.

7. Only one record description entry for the file must describe `rec-key` or the segments of `seg-key`. The Prime Record Key has the same character positions in every record of the file.

8. If the associated file connector is an external file connector, all File Description entries in the run unit that are associated with that file connector must define the same data description entry for `rec-key` or the segments of `seg-key` with the same relative location within the record.

9. Additional Reference

**ALTERNATE RECORD KEY**

**REDEFINES**

**Function**

The REDEFINES clause allows different data description entries to describe the same storage area.

**General Format**

```
level-number [data-name FILLER] REDEFINES other-data-item
```

[other-data-item]

is a data-name. It identifies the data description entry that first defines the storage area.

**Note**

Level-number, data-name, and FILLER are not part of the REDEFINES clause. They are included in the general format only to clarify the relative position of the clause.
Syntax Rules

1. The subject of the REDEFINES clause is the data-name or FILLER in a Format 1 data description entry.

2. The REDEFINES clause must immediately follow its subject.

3. The level-numbers of the subject of the REDEFINES clause and other-data-item must be the same. However, they cannot be either 66 or 88.

4. The REDEFINES clause cannot be used in a level 01 entry in the File Section.

5. The data description entry for other-data-item cannot contain an OCCURS clause. However, other-data-item can be subordinate to an item whose data description entry contains an OCCURS clause. In that case, the reference to other-data-item in the REDEFINES clause cannot be subscripted or indexed.

6. Neither the original definition nor the redefinition can contain a variable occurrence data item.

7. If other-data-item is either an external record or anything other than a level 01 entry, the number of character positions it contains must be greater than or equal to the number in the subject of the REDEFINES clause. If other-data-item is a level 01 entry, and is not an external record, its description need not follow this rule; that is, other-data-item can contain fewer character positions than the subject of the REDEFINES clause.

8. Other-data-item cannot be qualified even if it is not unique. The reference to other-data-item is unique without qualification because of the placement of the REDEFINES clause.

9. A program can have multiple redefinitions of the same character positions. However, they must all refer to other-data-item, the data-name that originally defined the area.

10. The redefining entries cannot contain VALUE clauses except in condition-name entries.

11. No entry with a level-number lower than that of other-data-item can occur between the data description entry for other-data-item and the redefinition.

12. The entries redefining the storage area must immediately follow those that originally defined it. There can be no intervening entries that define additional storage areas.

General Rules

1. Storage allocation starts at the location of other-data-item. Storage allocation continues until it defines the number of character positions in the data item referred to by the subject of the REDEFINES clause.

2. If more than one data description entry defines the same character position, the program can refer to the character position using the data-name associated with any of those data description entries.

Additional References

- Data Description entry
- Section 5.2.3: Additional Alignment Rules for Record Allocation

Example

This example shows the following:

- A sample program containing multiple redefinitions of the same area
- The results of the sample program statements
- The allowable subscripts and the contents for each data item in the program
IDENTIFICATION DIVISION.
PROGRAM-ID. REDEFINES-TEST.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ITEMS.
   03 FILLER PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
   03 FILLER PIC X(10) VALUE "0123456789".
   03 ITEMS OCCURS 36 TIMES PIC X.
01 REDEFINES ITEM.
   03 ITEM.
      05 ITEM OCCURS 26 TIMES PIC X.
   03 REDEFINES ITEMT.
      05 ITEMT OCCURS 13 TIMES.
      07 ITEMT PIC X.
      09 ITEMT PIC X.
09 ITEMS PIC X.
03 ITEM.
   05 ITEM OCCURS 5 TIMES PIC XX.
PROCEDURES DIVISION.
XXX.

DISPLAY ITEM.
   ABCDEFGHJKLmnopqrstuvwxyz0123456789
2  x
2  0
2  9
2  ABCDEFGHJKLmnopqrstuvwxyz0123456789
2  2
2  A8
2  Y2
2  AB
2  Y2
2  V
2  R
2  Z
2  0123456789
01  01
09  09

STOP RUN.

RESULTS
## REDEFINES Clause

<table>
<thead>
<tr>
<th>Name</th>
<th>Item Contents and Subscript Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEM</td>
<td>ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789</td>
</tr>
<tr>
<td>ITEMB</td>
<td>ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789</td>
</tr>
<tr>
<td>ITEMC</td>
<td>ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789</td>
</tr>
<tr>
<td>ITEMD</td>
<td>ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789</td>
</tr>
<tr>
<td>ItemE</td>
<td>ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789</td>
</tr>
<tr>
<td>ItemF</td>
<td>ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789</td>
</tr>
<tr>
<td>ItemG</td>
<td>ACEGIKMOPQRSTUVWXYZ0123456789</td>
</tr>
<tr>
<td>ItemH</td>
<td>BDFHJLNPQRSTUVWXYZ0123456789</td>
</tr>
<tr>
<td>ItemI</td>
<td>0123456789</td>
</tr>
<tr>
<td>ItemJ</td>
<td>0123456789</td>
</tr>
</tbody>
</table>

* Subscript Not Applicable

## RENAMES

### Function

The RENAMES clause groups elementary items in alternative or overlapping ways.
Data Division

General Format

```
66 new-name RENAMES rename-start {THRU THROUGH} rename-end
```

[new-name]
is the data-name of the item being described. It identifies an alternate grouping of one or more items in a record.

[rename-start]
is the data-name of the leftmost data item in the area. It can be qualified.

[rename-end]
is the data-name of the rightmost data item in the area. It can be qualified.

Note

Level-number 66 and new-name are not part of the RENAMES clause. They are in the general format only to clarify the relationship.

Syntax Rules

1. A logical record can have any number of RENAMES entries.

2. All RENAMES entries referring to data items in a logical record must immediately follow the last data description entry of the record description entry.

3. The program cannot qualify data-names with new-name.

4. The program can qualify new-name only by the names of the associated level 01, FD, or SD entries.

5. The data description entries for rename-start and rename-end:
   - Cannot have an OCCURS clause
   - Cannot be subordinate to an item whose data description entry has an OCCURS clause

6. rename-start and rename-end must be the names of elementary items or groups of elementary items in the same logical record. They cannot be the same data-name.

7. A level 66 entry cannot rename another level 66 entry. Nor can it rename a level 88, level 01, or level 77 entry.

8. None of the items in the range, including rename-start and rename-end, can be variable occurrence data items.

9. The words THRU and THROUGH are equivalent.

10. rename-end cannot be subordinate to rename-start. The beginning of rename-end cannot be to the left of the beginning of rename-start. The end of rename-end must be to the right of the end of rename-start.

General Rules

1. If rename-end is used, new-name includes all elementary items:
   - Starting with rename-start, if rename-start is an elementary item or the first elementary item in rename-start, or if rename-start is a group item
   - Ending with rename-end, if rename-end is an elementary item or the last elementary item in rename-end, or if rename-end is a group item
2. If `rename-end` is not used, all data attributes for `rename-start` become data attributes for `new-name`. In this case, you are renaming a single data item. If that item is a group item, `new-name` is also treated as a group item. If that item is an elementary item, `new-name` is also treated as an elementary item.

**Additional Reference**

Data Description

**Example**

In the following example, the box RESULTS displays the values given when using the RENAMES clause:

```assembly
REPORT

FUNCTION

REPORT

1. Each `report-name` in the REPORT clause must be the subject of a Report Description entry (RD) in the Report Section of the same program. `report-name` can appear in only one REPORT clause.

2. `report-names` can appear in any order.

3. The file-name in a file description entry for a Report File can be referenced only by the OPEN statement with the OUTPUT or EXTEND phrase or by the CLOSE statement.

**General Rules**

1. More than one `report-name` in a REPORT clause indicates that the file contains more than one report.
2. After executing an INITIATE statement and before executing a TERMINATE statement for the same report file, the report file is under the control of the Report Writer Control System (RWCS). While a report file is under control of the RWCS, no input/output statement may reference that report file.

3. If the associated file connector is an external file connector, every file description entry in the run unit associated with that file connector must describe it as a report file.

**Technical Note**

On OpenVMS, the DCL PRINT command inserts a form-feed character when a form is within four lines from the bottom. This positions the report to the top of the next logical page.

Report Writer files are written in print format. Line spacing positions the report to the top of the next logical page. Therefore, use the PRINT/NOFEED command to suppress the insertion of form-feed characters and to print your Report Writer files correctly. For example:

```
$ PRINT/NOFEED full-file-name
```

**Additional References**

- FD (File Description)
- RD (Report Description)

**REQUIRED (Alpha, I64)**

REQUIRED (Alpha, I64)

**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
```

**Syntax Rule**

The REQUIRED clause cannot be specified in the description of a literal screen item.

**General Rules**

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

2. The REQUIRED clause takes effect during the execution of any ACCEPT statement when the cursor enters the screen item. Until this clause is satisfied, the operator cannot leave the field and normal terminator keystrokes are rejected.

3. To satisfy this clause, alphanumeric screen items must contain at least one nonspace character, and numeric screen items must have a nonzero value.

4. 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.

5. The REQUIRED clause is not 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 field must always be filled entirely by the user.
7. The REQUIRED clause is ignored for an output field.

**Additional Reference**

ACCEPT statement in Chapter 6, *Procedure Division*

---

**REVERSE-VIDEO (Alpha, I64)**

**Function**

The REVERSE-VIDEO clause specifies that the field is displayed with the default or specified foreground and background colors exchanged.

**General Format**

```
REVERSE-VIDEO
```

**Syntax Rule**

The REVERSE-VIDEO clause can be specified only for elementary screen items.

**Additional Reference**

- ACCEPT statement in Chapter 6, *Procedure Division*
- DISPLAY statement in Chapter 6, *Procedure Division*

---

**SECURE (Alpha, I64)**

**Function**

The SECURE clause suppresses the display of input characters on the screen.

**General Format**

```
SECURE
```

**Syntax Rule**

The SECURE clause can only be specified for an input screen item.

**General Rules**

1. If the SECURE clause is specified at group level, it applies to each input screen item in that group.
2. When the SECURE clause is used, characters introduced for the input field do not appear on the screen, yet the cursor moves as usual.

**Additional Reference**

ACCEPT statement in Chapter 6, *Procedure Division*

---

**SIGN**

SIGN
Function

The SIGN clause specifies the operational sign's position and type of representation.

For screen description entries, the SIGN clause specifies the position of the sign character in the field. The sign character always occupies a separate position in the field, regardless of whether or not you specify SEPARATE.

General Formats

Format 1 (Data Description and Screen Description Entries)

```
[ SIGN IS ] { LEADING }
{ TRAILING } [ SEPARATE CHARACTER ]
```

Format 2 (Report Group Description Entries)

```
[ SIGN IS ] { LEADING }
{ TRAILING } SEPARATE CHARACTER
```

Syntax Rules

Format 1

1. The SIGN clause can be used only in a numeric data description entry or screen description entry whose PICTURE contains the S symbol, or for a group item containing such entries.

2. The data items to which the SIGN clause applies must have display usage.

3. If a file description entry has a CODE-SET clause, all signed numeric data description entries associated with the file description entry must contain the SIGN IS SEPARATE clause.

General Rules

Both Formats

1. The SIGN clause specifies the operational sign's position and type of representation. It applies to a numeric data description entry or screen description entry or to each numeric data description entry or screen description entry subordinate to a group.

2. The SIGN clause applies only to numeric data description entries or screen description entries whose PICTURE clause contains the S symbol. S indicates the presence of an operational sign. However, S does not specify the sign's representation or, necessarily, its position.

3. If you specify the SIGN clause for both a group item and a group item subordinate to it, the SIGN clause for the subordinate group overrides the group item SIGN clause.

4. If you specify the SIGN clause for both a group item and an elementary numeric item subordinate to it, the SIGN clause for the elementary item overrides the group item SIGN clause.

5. A numeric data description entry or screen description entry to which no optional SIGN clause applies, but whose PICTURE contains an S symbol, has an operational sign.
   - The numeric data description entry is equivalent to an entry that contains the SIGN IS TRAILING clause without the SEPARATE CHARACTER phrase.
   - The screen description entry is equivalent to an entry that contains the SIGN IS TRAILING with the SEPARATE CHARACTER phrase.

6. If you specify the SEPARATE CHARACTER phrase (or it is implied):
   - The operational sign is the leading (or trailing) character of the elementary numeric data item. The sign does not share this position with a digit.
• The S symbol in the PICTURE counts toward data or screen item size. That is, it represents a character position.

• The operational sign for positive is the plus sign (+).

• The operational sign for negative is the minus sign (-).

7. Every numeric data item whose PICTURE contains the S symbol is a signed numeric data item. If you specify the SIGN clause for such an item, necessary conversions for computations or comparisons occur automatically.

**Format 1 (Data Description)**

1. If you do not specify the SEPARATE CHARACTER phrase:

   • The operational sign is associated with the leading (or trailing) digit position of the elementary numeric item. The sign shares this character position with a digit.

   • The S symbol in the PICTURE does not count toward the size of the item. That is, it does not represent a character position.

   • The character in the operational sign position represents both a numeric digit and the item's algebraic sign. Table 5.11, “Positive and Negative Signs for All Numeric Digits” shows the characters representing positive and negative signs for all numeric digits. Where more than one character appears, the first is the character generated as the result of machine operations.

   **Table 5.11. Positive and Negative Signs for All Numeric Digits**

<table>
<thead>
<tr>
<th>Digit Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

**SOURCE**

**SOURCE**

**Function**

The SOURCE clause identifies a data item to be sent to an associated printable item defined within a Report Group Description entry.

**General Format**

SOURCE IS .source-id

[source-id]
names an elementary item in the Data Division.

**Syntax Rules**

1. If `source-id` is a Report Section item it must be either:
   - A PAGE-COUNTER
   - A LINE-COUNTER
   - A sum counter that is part of the report within which the SOURCE clause appears

2. The Report Writer Control System (RWCS) moves the contents of `source-id` to the printable item. `source-id` definitions must conform to the rules for sending items in the MOVE statement.

**General Rule**

The RWCS executes implicit MOVE statements specified by the SOURCE clauses when it formats the print lines (just before it presents them).

**Additional References**

- COLUMN NUMBER clause (printable item)
- TYPE clause
- MOVE statement in Chapter 6, *Procedure Division*

**SUM**

**SUM Function**

The SUM clause establishes a Report Writer sum counter and names the data items to be summed.

**General Format**

```
{ SUM { sum-name } ... [ UPON [ detail-report-group-name ] ... ] } ... 

[ RESET ON \{ control-foot-name \} \{ FINAL \} ]
```

=sum-name

names a numeric data item with an optional sign in the Subschema, File, Working-Storage, or Linkage Sections, or another sum counter in the Report Section.

detail-report-group-name

names a DETAIL report group.

control-foot-name

must reference a control-name in the report's CONTROL clause.
Data Division

Syntax Rules

1. A SUM clause can appear only in the description of a CONTROL FOOTING report group.

2. If there is no UPON phrase, any sum-name in the SUM clause that is itself a sum counter must be defined either in the same report group that contains this SUM clause or in a report group at a lower level in the control hierarchy of this report.

   If there is an UPON phrase, sum-name must not reference a sum counter.

3. If the associated report file connector is an external file connector and if sum-name references a numeric data item in the Subschema, File, Working-Storage, or Linkage Sections, then sum-name must reference the same external data item in all programs in the run unit.

4. detail-report-group-name must be a control-name in a CONTROL clause and must be the name of a DETAIL report group described in the same report as the CONTROL FOOTING report group in which the SUM clause appears.

5. detail-report-group-name may be qualified by a report-name.

6. control-foot-name must not be at a lower control level than the associated control level for the report group in which the RESET phrase appears.

   If FINAL appears in the RESET phrase, FINAL must also appear in the CONTROL clause for this report.

7. The highest permissible qualifier for sum-name is the report-name.

General Rules

1. The SUM clause establishes a sum counter. At run time, the Report Writer Control System (RWCS) adds the value in each sum-name to the sum counter. This addition is consistent with the rules for arithmetic statements.

2. The UPON phrase provides for selective subtotalling. Subtotalling occurs each time the RWCS processes the DETAIL report group referenced by detail-report-group-name.

3. If there is a RESET phrase, the RWCS will set the sum counter to zero when the RWCS is processing the designated level of control hierarchy. If there is no RESET phrase, the RWCS will set the sum counter in the CONTROL FOOTING report group to zero when the RWCS processes that report group.

   The RWCS initially sets sum counters to zero during the execution of the INITIATE statement for the report containing the sum counter.

4. The size of the sum counter is equal to the number of receiving character positions defined in the PICTURE clause that accompanies the SUM clause in the description of the elementary item.

5. Only one sum counter exists for an elementary report entry, regardless of the number of SUM clauses specified in the elementary report entry.

6. If the elementary report entry for a printable item contains a SUM clause, the sum counter serves as a source data item. On a control break, the RWCS moves the data from the sum counter to the printable item for presentation according to the rules of the MOVE statement.

7. If a data-name appears as the subject of an elementary report entry that contains a SUM clause, the data-name is the name of the sum counter; the data-name is not the name of a printable item that the entry may also define.

8. Procedure Division statements can alter the contents of sum counters.

9. During the execution of GENERATE and TERMINATE statements, the RWCS adds the values in sum-name to a sum counter.
10. The RWCS adds each individual sum-name into the sum counter when it processes the CONTROL FOOTING report group defining the sum counter.

**Technical Notes**

- The three categories of sum counter accumulation are as follows:
  - Subtotalling
  - Crossfooting
  - Rolling forward

  Subtotalling occurs only during execution of GENERATE statements and after any control break processing but before processing of the DETAIL report group. Crossfooting and rolling forward occur during the processing of CONTROL FOOTING report groups.

  - Subtotalling accumulates numeric data fields (sum-names) into a sum counter. sum-name must not reference a sum counter when subtotalling. If the SUM clause contains the UPON phrase, sum-names are subtotalled when a GENERATE statement executes for a DETAIL report group. If there is no UPON phrase, sum-names are subtotalled when any GENERATE data-name statement is executed for the report in which the SUM clause appears.

  - Crossfooting accumulates sum counters (sum-name) from the same CONTROL FOOTING report group into another sum counter. It is a horizontal sum of sums.

  Crossfooting occurs when a control break takes place and when the CONTROL FOOTING report group is processed.

  Crossfooting is performed according to the sequence in which sum counters are defined within the CONTROL FOOTING report group. That is, all crossfooting into the first sum counter defined in the CONTROL FOOTING report group is completed, and then all crossfooting into the second sum counter defined in the CONTROL FOOTING report group is completed. This procedure repeats until all crossfooting operations are completed.

  When one of the sum-names is the sum counter defined by the Data Description entry in which that sum clause appears, the initial value of that sum counter is used in the summing operation.

  - Rolling forward accumulates sum counters (sum-name) defined in lower level CONTROL FOOTING report groups into another sum counter. It is a vertical sum of sums. A sum counter in a lower level CONTROL FOOTING report group is rolled forward when a control break occurs and at the time the lower level CONTROL FOOTING report group is processed.

  - If two or more sum-names specify the same sum counter, then the sum counter is added as many times as the sum counter is referenced in the SUM clause. It is permissible for two or more of the sum-names to specify the same DETAIL report group. When a GENERATE data-name statement for such a DETAIL report group is given, the incrementing occurs repeatedly, as many times as the sum-name appears in the UPON phrase.

**Additional References**

- GENERATE statement in Chapter 6, Procedure Division
- TYPE clause
- Section 6.6.1: Arithmetic Operations
- Section 6.6.7: Overlapping Operands and Incompatible Data

**SYNCHRONIZED**
Function

The SYNCHRONIZED clause specifies elementary item alignment on word boundary offsets relative to a record's beginning. These offsets are related to the size and usage of the item being stored.

Syntax Rules

1. SYNC is an abbreviation for SYNCHRONIZED.

2. The SYNCHRONIZED clause can be used only for an elementary item.

General Rules

1. The SYNCHRONIZED clause aligns a data item in a record so that no other data item occupies any character positions between the required boundaries to the left and right of the data item.

2. If the number of character positions needed to store the data item is less than the number of positions between the required boundaries, no other data items occupy the unused positions. However, the unused character positions are included in the size of those group items:

   • To which the elementary item belongs
   • In which the elementary item is not the first subordinate item

   The first elementary item in a group item always aligns on the same boundary as the group item. In this case, any unused character positions do not affect the size of that group item.

3. The size of a SYNCHRONIZED data item equals the number of character positions between its natural boundaries. Therefore, the LEFT and RIGHT phrases have the same effect; they are equivalent to each other, and to the SYNCHRONIZED clause with neither the LEFT nor RIGHT phrases.

4. The SYNCHRONIZED clause does not change the size or operational sign position of the data item it specifies.

5. Each occurrence of the data item is synchronized if the clause applies to a data item whose data description entry also has an OCCURS clause, or to a data item subordinate to another data item whose data description entry has an OCCURS clause.

Technical Notes

• The SYNCHRONIZED clause does not affect the alignment of DISPLAY data items.

• The SYNCHRONIZED clause explicitly aligns COMP, COMP-1, COMP-2, POINTER, and INDEX data items on boundaries that are related to the size of the item.

   One word COMP items are aligned on 2-byte boundaries, longword items on 4-byte boundaries, and quadword items on 8-byte boundaries. All boundaries are relative to the beginning of the record containing the data item.

• The following table shows the alignment for each data type that the SYNCHRONIZED clause affects:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP (1 to 4 digits)</td>
<td>2-byte</td>
</tr>
<tr>
<td>COMP (5 to 9 digits)</td>
<td>4-byte</td>
</tr>
</tbody>
</table>
### Additional Reference

Section 5.2.3: Additional Alignment Rules for Record Allocation

## TYPE

### Function

The TYPE clause identifies the report group type and indicates when the Report Writer Control System (RWCS) is to process it.

### General Format

```
TYPE IS
  \{ REPORT HEADING
     RH
   \}
  \{ PAGE HEADING
     PH
   \}
  \{ CONTROL HEADING
     CH
   \}
  \{ DETAIL
     DE
   \}
  \{ CONTROL FOOTING
     CF
   \}
  \{ PAGE FOOTING
     PF
   \}
  \{ REPORT FOOTING
     RF
   \}

[control-head-name]

names a control-name in the CONTROL clause.

[control-foot-name]

names a control-name in the CONTROL clause.

### Syntax Rules

1. RH is an abbreviation for REPORT HEADING.
Data Division

PH is an abbreviation for PAGE HEADING.

CH is an abbreviation for CONTROL HEADING.

DE is an abbreviation for DETAIL.

CF is an abbreviation for CONTROL FOOTING.

PF is an abbreviation for PAGE FOOTING.

RF is an abbreviation for REPORT FOOTING.

2. These report groups may appear no more than once in the description of a report:
   - REPORT HEADING
   - PAGE HEADING
   - CONTROL HEADING FINAL
   - CONTROL FOOTING FINAL
   - PAGE FOOTING
   - REPORT FOOTING

3. The TYPE DETAIL report group may appear more than once in the description of a report.

4. If the TYPE clause specifies a CONTROL HEADING or CONTROL FOOTING report group, the *control-head-name*, *control-foot-name*, or FINAL entries must be specified in the CONTROL clause of the corresponding Report Description entry. For each control-name or FINAL phrase in the CONTROL clause of a Report Description entry, you can specify one CONTROL HEADING report group and one CONTROL FOOTING report group. However, the RWCS does not require either a CONTROL HEADING report group or a CONTROL FOOTING report group for each control-name or FINAL phrase in the CONTROL clause of a Report Description.

5. PAGE HEADING and PAGE FOOTING report groups may appear only if the corresponding Report Description entry specifies a PAGE clause.

6. In CONTROL FOOTING, PAGE HEADING, PAGE FOOTING, and REPORT FOOTING report groups, SOURCE clauses and USE statements must not reference any of the following:
   - Formats 2 and 3 Report Group Description data items containing a control data item
   - Data items subordinate to a control data item
   - A redefinition or renaming of any part of a control data item

7. In PAGE HEADING and PAGE FOOTING report groups, SOURCE clauses and USE statements must not reference either *control-head-name* or *control-foot-name*.

8. When the Procedure Division specifies a GENERATE report-name statement, the corresponding Report Description entry must define no more than one DETAIL report group. If there are no GENERATE group-data-name statements in the Procedure Division, the RWCS does not require a DETAIL report group. If there are multiple TYPE DETAIL report groups in the report, the GENERATE group-data-name statement must be used.

**General Rules**

1. The Report Writer Control System (RWCS) processes DETAIL report groups as a direct result of the GENERATE statement. If a report group specified in the GENERATE statement is not a TYPE DETAIL report
group, a summary report is produced. If a report group specified in the GENERATE statement is a TYPE DETAIL report group, a detailed report is produced.

2. The RWCS executes the following procedures (a to f) when it processes a DETAIL report group in response to a GENERATE statement.

When the description of a report includes exactly one DETAIL report group, the detail-related processing that the RWCS executes in response to a GENERATE report-name statement is described in procedures a to d. The RWCS performs these procedures as though a GENERATE group-data-name statement were being executed.

When the description of a report includes no DETAIL report groups, the detail-related processing that the RWCS executes in response to a GENERATE report-name statement is described in procedures a and b. These procedures are performed as though the description of the report included exactly one DETAIL report group, and a GENERATE detail-report-group statement were being executed.

a. The RWCS performs any control break processing.

b. The RWCS performs any subtotalling that has been designated for the DETAIL report group.

c. If there is a USE BEFORE REPORTING procedure referring to the data-name of the report group, the RWCS executes the USE procedure.

d. If a SUPPRESS statement has been executed, or if the report group is not printable, no further processing is done for the report group.

e. If the RWCS processes a DETAIL report group as a consequence of the GENERATE report-name statement, no further processing is done for the report group.

f. If neither procedure d nor procedure e applies, the RWCS formats the print lines and presents the DETAIL report group.

3. To detect and trigger control breaks for a specific report, the RWCS:

a. Establishes the initial values of control data items as the prior values when the INITIATE statement executes.

b. Compares the prior values to the current values of control data items when a GENERATE statement executes. If the current values do not compare to the prior values, a control break occurs. If a control break occurs, the current values are saved as prior values and steps c, d, and e are performed.

c. Presents the CONTROL FOOTING and CONTROL HEADING report groups associated with the control break. The CONTROL FOOTING report groups presented are at a less major level than the level at which the control break occurred. The CONTROL HEADING report groups presented are in the order of major level to break level.

d. Processes any PAGE HEADING and PAGE FOOTING report groups when it must start a new page to present a CONTROL HEADING, DETAIL, or CONTROL FOOTING.

e. Repeats steps b, c, and d until the last control break is processed.

4. The prior values (refer to General Rule 3) may be referenced by the program:

- During the control break processing of a CONTROL FOOTING report group. Any references to control data items in a USE procedure or SOURCE clause associated with that CONTROL FOOTING report group are supplied with prior values.

- When a TERMINATE statement executes. The RWCS makes the prior values available to the SOURCE clause or the USE procedure references in CONTROL FOOTING report groups as though the control break had been detected in the highest control data-name.

- At the time the RWCS processes the report group. All other data item references within report groups and their USE procedures access the current values contained within the data items.
5. The RWCS presents the REPORT HEADING report group only once for each report, as the first report group of that report. It is processed when the first GENERATE statement is executed.

6. The RWCS presents the PAGE HEADING report group as the first report group on each page of the report, except for the following conditions:

   • A page containing only a REPORT HEADING report group.
   • A page containing only a REPORT FOOTING report group.
   • A page containing a REPORT HEADING report group that is not the only report group on the page. In this case, the PAGE HEADING report group is the second report group on the page.

7. The RWCS processes the CONTROL HEADING report group at the end of a control break for a specific control-head-name. The CONTROL HEADING FINAL report group is presented only once for each report, as the first body group (CONTROL HEADING, DETAIL, and CONTROL FOOTING) of that report. Other CONTROL HEADING report groups are presented when the RWCS detects a control break on the control-head-name during the execution of GENERATE statements. Control break processing for any CONTROL HEADING report group occurs with the highest control level of the break and includes all lower levels.

8. The RWCS presents CONTROL FOOTING report group at the beginning of a control break for a specific control-foot-name. The CONTROL FOOTING FINAL report group is presented only once for each report, as the last body group (CONTROL HEADING, DETAIL, and CONTROL FOOTING) of that report. If, during the execution of a GENERATE statement, the RWCS detects a control break, control break processing for any CONTROL FOOTING report group occurs with the highest control level of the break and includes all lower levels. Upon execution of the TERMINATE statement, the RWCS processes all CONTROL FOOTING report groups if the GENERATE statement has executed at least once.

9. The RWCS processes the PAGE FOOTING report group as the last report group on each page of the report, except for the following conditions:

   • A page containing only a REPORT HEADING report group.
   • A page containing only a REPORT FOOTING report group.
   • A page containing a REPORT FOOTING report group that is not to be the only report group on the page. In this case, the PAGE FOOTING report group is the second to the last report group on the page.

10. The RWCS processes the REPORT FOOTING report group, if defined, only once per report and as the last report group of that report. During the execution of a TERMINATE statement, the RWCS processes the corresponding REPORT FOOTING report group if at least one GENERATE statement is executed for the report.

11. The RWCS checks for these three conditions before it processes a REPORT HEADING, PAGE HEADING, CONTROL HEADING, PAGE FOOTING, or a REPORT FOOTING report group:

   • If there is a USE BEFORE REPORTING procedure referencing the data-name of the report group, the USE procedure executes.
   • If a SUPPRESS statement has been executed, or if the report group is not printable, there is no further processing for the report group.
   • If a SUPPRESS statement has not been executed and the report group is printable, the RWCS formats the print lines and presents the report group according to the presentation rules for that type of report group.

12. The RWCS executes the following procedures when it processes a CONTROL FOOTING report group.

   Control breaks occur during the processing of a GENERATE statement. The GENERATE rules specify that the RWCS produces the CONTROL FOOTING report groups beginning at the minor level, and proceeding
Data Division

upwards, through and including the highest control level. Although no CONTROL FOOTING report group has been defined for a given control data-name, the RWCS will still have to execute procedure 12f if a RESET phrase within the report description specifies that control data-name.

a. Sum counters are crossfooted. All sum counters defined in this report group that are operands of SUM clauses in the same report group are added to their sum counters.

b. Sum counters are rolled forward. All sum counters defined in the report group that are operands of SUM clauses in higher level CONTROL FOOTING report groups are added to the higher level sum counters.

c. If there is a USE BEFORE REPORTING group-data-name declarative procedure, the RWCS executes the USE procedure.

d. If a SUPPRESS statement has been executed, or if the report group is not printable, the RWCS executes procedure 12f.

e. If a suppress statement has not been executed and the report group is printable, the RWCS formats the print lines and presents the report group according to the presentation rules for CONTROL FOOTING report groups.

f. The RWCS resets those sum counters that are to be reset when the RWCS processes this level in the control hierarchy.

Additional References

• CONTROL clause
• Data-Name
• LINE NUMBER (Alpha, I64) clause (General Rule 4)
• SUM clause
• TERMINATE statement in Chapter 6, Procedure Division
• Appendix D: Report Writer Presentation Rules and Tables

UNDERLINE

Function

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

Syntax Rule

The UNDERLINE clause may be specified only for elementary screen items.
The USAGE clause specifies the internal format of a data item or screen item.

Syntax Rules

1. **BINARY** is a synonym for COMPUTATIONAL and COMP.

   On Alpha and I64 systems, except for restrictions on the PICTURE clause, COMPUTATIONAL-5 and COMPUTATIONAL-X are synonyms for COMPUTATIONAL and COMP.

2. **COMP** is an abbreviation for COMPUTATIONAL.

3. **COMP-1** is an abbreviation for COMPUTATIONAL-1.

4. 

5. **COMP-3** is an abbreviation for COMPUTATIONAL-3.

6. **PACKED-DECIMAL** is a synonym for COMPUTATIONAL-3 and COMP-3.

7. On Alpha and I64 systems, **COMP-5** is an abbreviation for COMPUTATIONAL-5.

8. 

9. On Alpha and I64 systems, **FLOAT-SHORT** is a synonym for COMPUTATIONAL-1.

10. On Alpha and I64 systems, **FLOAT-LONG** and **FLOAT-EXTENDED** are synonyms for COMPUTATIONAL-2.

11. You can use the USAGE clause in any data description entry with a level-number other than 66 or 88.

12. If the USAGE clause is in the data description for a group item, it can also be in data description entries for subordinate elementary and group items. However, the usage of a subordinate item must be the same as that in the group item data description entry.

13. The PICTURE character-string of a COMP or COMP-3 item can contain only the following symbols:

   - 9
• S
• V
• P

14. On Alpha and I64 systems, the PICTURE character-string of a COMP-5 or COMP-X item can contain only the following symbols:
• 9
• S
• X (but not in combination with 9 or S)

15. An index data item reference can appear in only:
• A SEARCH or SET statement
• A relation condition
• The USING phrase of the Procedure Division header
• The USING phrase of the CALL statement

16. A report description entry or a screen description entry can only specify USAGE IS DISPLAY.

17. The data description entry for a USAGE IS INDEX data item cannot contain any of the following clauses:
• BLANK WHEN ZERO
• JUSTIFIED
• PICTURE
• VALUE IS

18. An elementary item with the USAGE IS INDEX clause cannot be a conditional variable; that is, the elementary item's value cannot be specified by level 88 items.

19. The data description entry of a BINARY-CHAR, BINARY-SHORT, BINARY-LONG, BINARY-DOUBLE, COMP-1, COMP-2, POINTER, or POINTER-64 item cannot have a PICTURE clause. However, they are numeric and signed.

20. The subject of a data description entry containing the USAGE IS POINTER clause must not include any of the following clauses:
• BLANK WHEN ZERO
• JUSTIFIED
• PICTURE

**General Rules**

1. You can specify the USAGE clause in the data description entry for a group item. In this case, it applies to each elementary item in the group. However, you cannot reference the group item in any operations that do not permit alphanumeric operands. See rules 4 and 8 for more information.

2. The USAGE clause specifies the representation of an elementary data item in storage. It does not affect the way that the program uses the item. However, the rules for some Procedure Division statements restrict the USAGE clause of statement operands.
3. A BINARY-CHAR, BINARY-SHORT, BINARY-LONG, BINARY-DOUBLE, COMP, COMP-1, COMP-2, COMP-3, COMP-5, COMP-X, FLOAT-SHORT, FLOAT-LONG, or FLOAT-EXTENDED item can represent a value used in computations. The PICTURE clauses for COMP and COMP-3 items must be numeric. The PICTURE clauses for COMP-5 and COMP-X items may be numeric or X.

4. A POINTER data item can represent an address value used in computations. The compiler internally treats this item as a binary integer. References to a POINTER item are allowed in the same context as references to a COMP integer.

5. If the data description entry for a group item specifies BINARY-CHAR, BINARY-SHORT, BINARY-LONG, BINARY-DOUBLE, COMP, COMP-1, COMP-2, COMP-3, COMP-5, COMP-X, FLOAT-SHORT, FLOAT-LONG, FLOAT-EXTENDED, POINTER, or POINTER-64 usage, the usage applies to elementary items in the group. It does not apply to the group itself; and the program cannot use the group item in computations.

6. The USAGE IS DISPLAY clause specifies that the data item is in Standard Data Format.

7. If no USAGE clause applies to an elementary item, its usage is DISPLAY.

8. If the USAGE IS INDEX clause applies to an elementary item, the elementary item is called an index data item. It contains a value that must correspond to an occurrence number of a table element.

9. If the data description entry for a group item specifies USAGE IS INDEX, all elementary items in the group are index data items. However, the group itself is not an index data item.

10. When a MOVE or input-output statement refers to a group that contains an index data item, the index data item is not converted to another format during the operation. Conversion will occur when the CONVERSION option is specified on ACCEPT or DISPLAY.

11. The USAGE IS POINTER clause can be used only in a File, Working-Storage, or Linkage Section data description entry.

12. On OpenVMS Alpha and I64 systems, the USAGE IS POINTER-64 clause is provided for limited use in interfacing with applications in languages requiring a 64-bit pointer. See Technical Notes.

**Technical Notes**

1. The way a data item is represented in the Data Division of a COBOL program determines whether it will be stored as an integer, floating-point, packed decimal, display numeric, or character string (text) data type. Tables 5.12 and 5.13 show the following:

   - COBOL data description entries and their corresponding data types
   - The allocated storage in bytes for each entry; allocated storage is the same on Tru64 UNIX and OpenVMS Alpha and I64 (except for POINTER)

   Table 5.12, “Unscaled Data Items, Allocated Storage, and Corresponding Data Types” gives the corresponding data types for unscaled data items, and Table 5.13, “Scaled Data Items, Allocated Storage, and Data Types” gives the data types for scaled data items.

   For example, a data item described as PIC S9(4) USAGE IS DISPLAY SIGN IS TRAILING is stored in 4 bytes of storage as a right overpunch value.

**Note**

The default USAGE for a data item is DISPLAY. Therefore, you do not need to specify the USAGE clause for display numeric, alphabetic, and alphanumeric data items.
### Table 5.12. Unscaled Data Items, Allocated Storage, and Corresponding Data Types

<table>
<thead>
<tr>
<th>PICTURE Clause</th>
<th>USAGE Clause</th>
<th>SIGN Clause</th>
<th>Allocated Storage in Bytes</th>
<th>Standard Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC 9(n)</td>
<td></td>
<td></td>
<td>n</td>
<td>Right (trailing) overpunch</td>
</tr>
<tr>
<td>[n &lt;= 31 (Alpha, I64)]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC 9(n)</td>
<td></td>
<td>TRAILING</td>
<td>n</td>
<td>Right (trailing) overpunch</td>
</tr>
<tr>
<td>[n &lt;= 31 (Alpha, I64)]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC 9(n)</td>
<td></td>
<td>LEADING</td>
<td>n</td>
<td>Left (leading) overpunch</td>
</tr>
<tr>
<td>[n &lt;= 31 (Alpha, I64)]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC 9(n)</td>
<td></td>
<td>TRAILING SEPARATE</td>
<td>n+1</td>
<td>Right (trailing) separate</td>
</tr>
<tr>
<td>[n &lt;= 31 (Alpha, I64)]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC 9(n)</td>
<td></td>
<td>LEADING SEPARATE</td>
<td>n+1</td>
<td>Left (leading) separate</td>
</tr>
<tr>
<td>[n &lt;= 31 (Alpha, I64)]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC 9(n)</td>
<td></td>
<td></td>
<td>n</td>
<td>Unscaled numeric</td>
</tr>
<tr>
<td>[n &lt;= 31 (Alpha, I64)]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC 9(n)</td>
<td>COMP</td>
<td></td>
<td>2</td>
<td>Word integer$^\dagger$</td>
</tr>
<tr>
<td>[n &lt;= 4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC 9(n)</td>
<td>COMP</td>
<td></td>
<td>4</td>
<td>Longword integer$^\dagger$</td>
</tr>
<tr>
<td>[5 &lt;= n &lt;= 9]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC 9(n)</td>
<td>COMP</td>
<td></td>
<td>8</td>
<td>Quadword integer$^\dagger$</td>
</tr>
<tr>
<td>[n &lt;= 9]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PICTURE Clause</td>
<td>USAGE Clause</td>
<td>SIGN Clause</td>
<td>Allocated Storage in Bytes</td>
<td>Standard Data Type</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------</td>
<td>-------------</td>
<td>----------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>([10 \leq n \leq 18])</td>
<td>COMP-5 (Alpha, I64) COMP-X (Alpha, I64)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC 9(n) ([19 \leq n \leq 31]) (Alpha, I64)</td>
<td>COMP COMP-5 COMP-X</td>
<td>16</td>
<td>Octaword integer (Alpha, I64)</td>
<td></td>
</tr>
<tr>
<td>PIC S9(n) ([n \leq 4])</td>
<td>COMP COMP-5 (Alpha, I64) COMP-X (Alpha, I64)</td>
<td>2</td>
<td>Word integer</td>
<td></td>
</tr>
<tr>
<td>PIC S9(n) ([5 \leq n \leq 9])</td>
<td>COMP COMP-5 (Alpha, I64) COMP-X (Alpha, I64)</td>
<td>4</td>
<td>Longword integer</td>
<td></td>
</tr>
<tr>
<td>PIC S9(n) ([10 \leq n \leq 18])</td>
<td>COMP COMP-5 (Alpha, I64) COMP-X (Alpha, I64)</td>
<td>8</td>
<td>Quadword integer</td>
<td></td>
</tr>
<tr>
<td>PIC S9(n) ([19 \leq n \leq 31]) (Alpha, I64)</td>
<td>COMP COMP-5 COMP-X</td>
<td>16</td>
<td>Octaword integer (Alpha, I64)</td>
<td></td>
</tr>
<tr>
<td>PIC X(n) ([n \leq 2]) (Alpha, I64)</td>
<td>COMP-5 COMP-X</td>
<td>2</td>
<td>Word integer (Alpha, I64)</td>
<td></td>
</tr>
<tr>
<td>PIC X(n) ([3 \leq n \leq 4]) (Alpha, I64)</td>
<td>COMP-5 COMP-X</td>
<td>4</td>
<td>Longword integer (Alpha, I64)</td>
<td></td>
</tr>
<tr>
<td>PICTURE Clause</td>
<td>USAGE Clause</td>
<td>SIGN Clause</td>
<td>Allocated Storage in Bytes</td>
<td>Standard Data Type</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>-------------</td>
<td>-----------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>PIC X(n) [5 &lt;= n &lt;= 8] (Alpha, I64)</td>
<td>COMP-5 COMP-X</td>
<td>8</td>
<td>Quadword integer (^1) (Alpha, I64)</td>
<td></td>
</tr>
<tr>
<td>Not applicable</td>
<td>INDEX</td>
<td>4</td>
<td>Longword integer</td>
<td></td>
</tr>
<tr>
<td>Not applicable</td>
<td>POINTER</td>
<td>4</td>
<td>Longword integer (OpenVMS)</td>
<td></td>
</tr>
<tr>
<td>POINTER</td>
<td>8</td>
<td>Quadword integer (Tru64 UNIX)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POINTER-64 (Alpha, I64)</td>
<td>8</td>
<td>Quadword integer (OpenVMS Alpha, I64)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not applicable</td>
<td>BINARY-CHAR UNSIGNED (Alpha, I64)</td>
<td>2</td>
<td>Word integer (^1) (Alpha, I64)</td>
<td></td>
</tr>
<tr>
<td>Not applicable</td>
<td>BINARY-SHORT UNSIGNED (Alpha, I64)</td>
<td>2</td>
<td>Word integer (^1) (Alpha, I64)</td>
<td></td>
</tr>
<tr>
<td>Not applicable</td>
<td>BINARY-LONG UNSIGNED (Alpha, I64)</td>
<td>4</td>
<td>Longword integer (^1) (Alpha, I64)</td>
<td></td>
</tr>
<tr>
<td>Not applicable</td>
<td>BINARY-DOUBLE UNSIGNED (Alpha, I64)</td>
<td>8</td>
<td>Quadword integer (^1) (Alpha, I64)</td>
<td></td>
</tr>
<tr>
<td>Not applicable</td>
<td>BINARY-CHAR SIGNED (Alpha, I64) BINARY-CHAR (Alpha, I64)</td>
<td>2</td>
<td>Word integer (Alpha, I64)</td>
<td></td>
</tr>
<tr>
<td>Not applicable</td>
<td>BINARY-SHORT SIGNED (Alpha, I64) BINARY-SHORT (Alpha, I64)</td>
<td>2</td>
<td>Word integer (Alpha, I64)</td>
<td></td>
</tr>
<tr>
<td>Not applicable</td>
<td>BINARY-LONG SIGNED (Alpha, I64)</td>
<td>4</td>
<td>Longword integer (Alpha, I64)</td>
<td></td>
</tr>
<tr>
<td>PICTURE Clause</td>
<td>USAGE Clause</td>
<td>SIGN Clause</td>
<td>Allocated Storage in Bytes</td>
<td>Standard Data Type</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>-------------</td>
<td>-----------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>BINARY-LONG (Alpha, I64)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not applicable</td>
<td>BINARY-DOUBLE SIGNED (Alpha, I64)</td>
<td>BINARY-DOUBLE (Alpha, I64)</td>
<td>8</td>
<td>Quadword integer (Alpha, I64)</td>
</tr>
<tr>
<td>Not applicable</td>
<td>COMP-1</td>
<td></td>
<td>4</td>
<td>F_floating S_format 2 (Alpha, I64)</td>
</tr>
<tr>
<td>Not applicable</td>
<td>COMP-2</td>
<td></td>
<td>8</td>
<td>D_floating G_floating (Alpha, I64) T_format 2 (Alpha, I64)</td>
</tr>
<tr>
<td>PIC 9(n) [n &lt;= 31 (Alpha, I64)]</td>
<td>COMP-3</td>
<td></td>
<td>(n+1)/2 rounded up</td>
<td>Packed decimal 1</td>
</tr>
<tr>
<td>PIC S9(n) [n &lt;= 31 (Alpha, I64)]</td>
<td>COMP-3</td>
<td></td>
<td>(n+1)/2 rounded up</td>
<td>Packed decimal</td>
</tr>
<tr>
<td>PIC X(n) [n &lt;= 268,435,455]</td>
<td>DISPLAY</td>
<td></td>
<td>n</td>
<td>ASCII text</td>
</tr>
<tr>
<td>PIC A(n) [n &lt;= 268,435,455]</td>
<td>DISPLAY</td>
<td></td>
<td>n</td>
<td>ASCII text</td>
</tr>
</tbody>
</table>

1 The generated code treats this data type as a positive value in all contexts except when it is a receiving-field operand. In this case, the compiler stores the absolute value of the source data item.

2 On OpenVMS Alpha and I64 systems, the data type depends on the /FLOAT qualifier. On Tru64 UNIX systems, it is always S format for COMP-1, and always T format for COMP-2. Refer to the VSI COBOL User Manual Appendix B, describing compatibility with VSI COBOL for information on the /FLOAT=IEEE qualifier.

Table 5.13. Scaled Data Items, Allocated Storage, and Data Types

<table>
<thead>
<tr>
<th>PICTURE Clause</th>
<th>USAGE Clause</th>
<th>SIGN Clause</th>
<th>Allocated Storage in Bytes</th>
<th>Standard Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC S9(n)V9(s)</td>
<td>DISPLAY</td>
<td></td>
<td>n+s</td>
<td>Right (trailing)</td>
</tr>
<tr>
<td>PICTURE Clause</td>
<td>USAGE Clause</td>
<td>SIGN Clause</td>
<td>Allocated Storage in Bytes</td>
<td>Standard Data Type</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------</td>
<td>-------------</td>
<td>-----------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>([(n+s) \leq 31 (\text{Alpha, I64})])</td>
<td></td>
<td></td>
<td></td>
<td>overpunch</td>
</tr>
<tr>
<td>PIC S9(n)V9(s) ([(n+s) \leq 31 (\text{Alpha, I64})])</td>
<td>DISPLAY</td>
<td>TRAILING</td>
<td>n+s</td>
<td>Right (trailing) overpunch</td>
</tr>
<tr>
<td>PIC S9(n)V9(s) ([(n+s) \leq 31 (\text{Alpha, I64})])</td>
<td>DISPLAY</td>
<td>LEADING</td>
<td>n+s</td>
<td>Left (leading) overpunch</td>
</tr>
<tr>
<td>PIC S9(n)V9(s) ([(n+s) \leq 31 (\text{Alpha, I64})])</td>
<td>DISPLAY</td>
<td>TRAILING</td>
<td>n+s+1</td>
<td>Right (trailing) separate</td>
</tr>
<tr>
<td>PIC S9(n)V9(s) ([(n+s) \leq 31 (\text{Alpha, I64})])</td>
<td>DISPLAY</td>
<td>LEADING</td>
<td>n+s+1</td>
<td>Left (leading) separate</td>
</tr>
<tr>
<td>PIC 9(n)V9(s) ([(n+s) \leq 31 (\text{Alpha, I64})])</td>
<td>DISPLAY</td>
<td></td>
<td>n+s</td>
<td>Unsigned numeric</td>
</tr>
<tr>
<td>PIC 9(n)V9(s) ([(n+s) \leq 4])</td>
<td>COMP</td>
<td></td>
<td>2</td>
<td>Word integer (^1)</td>
</tr>
<tr>
<td>PIC 9(n)V9(s) ([5 \leq (n+s) \leq 9])</td>
<td>COMP</td>
<td></td>
<td>4</td>
<td>Longword integer (^1)</td>
</tr>
<tr>
<td>PIC 9(n)V9(s) ([10 \leq (n+s) \leq 18])</td>
<td>COMP</td>
<td></td>
<td>8</td>
<td>Quadword integer (^1)</td>
</tr>
<tr>
<td>PIC 9(n)V9(s) ([19 \leq (n+s) \leq 31]) ((\text{Alpha, I64}))</td>
<td>COMP</td>
<td></td>
<td>16</td>
<td>Octaword integer (^1) ((\text{Alpha, I64}))</td>
</tr>
<tr>
<td>PIC S9(n)V9(s) ([(n+s) \leq 4])</td>
<td>COMP</td>
<td></td>
<td>2</td>
<td>Word integer</td>
</tr>
</tbody>
</table>
### Data Division

<table>
<thead>
<tr>
<th>PICTURE Clause</th>
<th>USAGE Clause</th>
<th>SIGN Clause</th>
<th>Allocated Storage in Bytes</th>
<th>Standard Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC S9(n)V9(s)</td>
<td>COMP</td>
<td></td>
<td>4</td>
<td>Longword integer</td>
</tr>
<tr>
<td>[5 &lt;= (n+s) &lt;= 9]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC S9(n)V9(s)</td>
<td>COMP</td>
<td></td>
<td>8</td>
<td>Quadword integer</td>
</tr>
<tr>
<td>[10 &lt;= (n+s) &lt;= 18]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC S9(n)V9(s)</td>
<td>COMP</td>
<td></td>
<td>16</td>
<td>Octaword integer (Alpha, I64)</td>
</tr>
<tr>
<td>[19 &lt;= (n+s) &lt;= 31]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC 9(n)V9(s)</td>
<td>COMP-3</td>
<td></td>
<td>(\frac{(n+s+1)}{2}) rounded up</td>
<td>Packed decimal(^1)</td>
</tr>
<tr>
<td>[(n+s) &lt;= 31 (Alpha, I64)]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC S9(n)V9(s)</td>
<td>COMP-3</td>
<td></td>
<td>(\frac{(n+s+1)}{2}) rounded up</td>
<td>Packed decimal</td>
</tr>
<tr>
<td>[(n+s) &lt;= 31 (Alpha, I64)]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^1\)The generated code treats this data type as a positive operand in all contexts except when it is a receiving-field operand. In this case, the compiler stores the absolute value of the data type.

2.

### Additional References

- PICTURE clause

### VALUE IS

**VALUE IS**

**Function**

The VALUE IS clause defines the values associated with condition-names, the initial value of Working-Storage Section data items, the value of Report Section printable items, the compile-time initialization of variables to the address of data, external constants, and the constant values of literal screen items.
Data Division

General Format

Format 1

```
VALUE IS lit
```

Format 2

```
88 condition-name

   VALUE IS
   VALUES ARE

   \{ EXTERNAL external-name
   \{ REFERENCE data-name
   \}
   \{ low-val
   \}
   \{ THRU
   \}
   \{ THROUGH
   \}
   \{ high-val
   \}
   \}
   \}
   …
```

Format 3

```
VALUE IS \{ REFERENCE data-name
\}
```

Format 4

```
VALUE IS EXTERNAL external-name
```

[lit]

is a numeric or nonnumeric literal. In a screen description entry, it is a nonnumeric literal.

[external-name]

names a COBOL link-time bound constant. It must define a word or longword integer value. See Technical Notes for more information.

[data-name]

names a data item in the File or Working-Storage or Subschema Section. data-name may be qualified.

[low-val]

is a numeric or nonnumeric literal. It is the lowest value in a range of values associated with a condition-name in a level 88 data description entry.

[high-val]

is a numeric or nonnumeric literal. It is the highest value in a range of values associated with a condition-name in a level 88 data description entry.

[numERIC-integer-lit]

is a positive numeric integer literal.

Syntax Rules

1. The words THRU and THROUGH are equivalent.

2. You must associate a signed numeric literal with either of the following:
   a. A data item that has a signed numeric PICTURE character-string
   b. A COMP-1 or COMP-2 data item

3. If you specify a numeric literal value:
Data Division

a. It must fall in the range of values defined by the data item's PICTURE clause.

b. It must not require truncation of nonzero digits; that is, it cannot have nonzero digits in positions represented by Ps in the item's PICTURE clause.

4. If you specify a nonnumeric literal value, it must not exceed the size defined by the data item's PICTURE clause.

5. The Format 1, 3, and 4 VALUE IS clause cannot be used in any entry that is part of the description or redefinition of an external data record.

6. The Format 3 VALUE IS clause is allowed only for an item containing the USAGE IS POINTER phrase.

7. The subject of the associated data description entry in a Format 4 VALUE IS clause must define a word or longword data item.

8. In a screen description entry, the VALUE clause can be specified only at the elementary level.

General Rules

1. The VALUE IS clause must be consistent with other clauses in the data description of both the item and all subordinate items. The following rules apply:

   • If the category of the item is numeric, all literals in the VALUE IS clause must be numeric. lit is aligned in the data item according to Standard Alignment Rule 1.

   • If the category of the item is alphabetic, alphanumeric, alphanumeric edited, or numeric edited, all VALUE IS clause literals must be nonnumeric. lit is aligned in the data item as if the data item were defined as alphanumeric. Editing characters in the PICTURE clause count toward data item size but have no effect on initialization. Therefore, if lit applies to an edited item, it must be in an edited form; Standard Alignment Rule 3 applies.

   • The BLANK WHEN ZERO clause does not directly affect initialization. However, the BLANK WHEN ZERO clause can change the category of the data item. If the category of the data item changes, the rules that apply change accordingly.

   • The JUSTIFIED clause does not affect initialization.

2. In the File Section, the VALUE IS clause can apply only to condition-name entries. That is, you can use the clause only for level 88 data items. In the Linkage Section, VALUE IS produces a warning for the other 88 data items.

3. Format 2 applies only to condition-name entries.

4. If a VALUE IS clause is specified in a data description entry that contains an OCCURS clause with a DEPENDING ON phrase, every occurrence of the associated data item is set to the maximum value.

   A data item is associated with a variable occurrence data item in any of the following cases:

   • It is a group data item that contains a variable occurrence data item.

   • It is a variable occurrence data item.

   • It is subordinate to a variable occurrence data item.

   If a VALUE IS clause is associated with the data item referenced by a DEPENDING ON phrase, that value is considered to be placed in the data item after the variable occurrence data item is initialized.

5. If a VALUE IS clause is specified in a data description entry that contains an OCCURS clause, or in an entry that is subordinate to an OCCURS clause, every occurrence of the associated data item is assigned the specified value. (This applies to General Formats 1, 3, and 4.)
## Condition-Name Rules for Format 2

1. The VALUE IS clause is required in a condition-name entry. The condition-name entry can contain only the condition-name itself and the VALUE IS clause.

2. The characteristics of a condition-name are implicitly the same as those of its conditional variable.

3. When using the EXTERNAL option, the associated conditional variable must be a word or longword COMP data item.

4. When using the REFERENCE option, the associated conditional variable must be POINTER usage.

5. If the THRU phrase is used, each low-val, external-name, and data-name must be less than the corresponding high-val, external-name, and data-name.

## Rules for Other Data Description Entries

1. A Working-Storage Section VALUE IS clause takes effect only when the program enters its initial state.

2. The VALUE IS clause initializes the data item to the value of lit.

3. If a data item's data description entry does not have a VALUE IS clause, the initial contents of the data item are the following:
   - Zero, for numeric items
   - Undefined, for index data items, and data items whose descriptions include or are subordinate to an OCCURS clause
   - Spaces, for all other items

4. In the Report Section, if an elementary report entry contains a VALUE IS clause but does not contain a GROUP INDICATE clause, the printable item assumes the specified value each time the Report Writer Control System (RWCS) prints the Report Group. However, if the entry contains the GROUP INDICATE clause, the RWCS presents the specified value only when certain run-time conditions exist. See the description of the GROUP INDICATE clause for more information.

5. The VALUE IS clause cannot be used in a data description entry that has a REDEFINES clause or is subordinate to a data description entry with a REDEFINES clause.

6. The VALUE IS clause can be in a data description entry for a group item. In this case:
   - lit must be a figurative constant or nonnumeric literal.
   - The group area is initialized as if the group were an elementary alphanumeric data item.
   - Initialization of group items is not affected by the characteristics of the group's subordinate group or elementary items.
   - The VALUE IS clause cannot be used in data description entries for the group's subordinate group or elementary items.

7. The VALUE IS clause cannot be used in the data description entry for a group that contains subordinate items with any of the following clauses:
   - JUSTIFIED
   - SYNCHRONIZED
   - USAGE (other than USAGE IS DISPLAY)

8. The VALUE IS clause cannot be used in the report group description entry for a group that contains subordinate items with a JUSTIFIED clause.
9. The Format 3 VALUE IS clause results in the compile-time initialization of its data description entry to the address of data-name or to numeric-integer-lit. Use this clause to pass arguments to non-COBOL procedures requiring an address rather than a user-defined word.

10. In Format 4, external-name must be the name of an external symbol (a symbol in another program unit) that is known to the linker when the program is linked.

11. The Format 4 VALUE IS clause results in the linker storing the value of external-name at the storage location defined by the data description entry containing the VALUE IS EXTERNAL clause.

**Technical Notes**

- external-name is a COBOL word formed according to the rules for user-defined names. The compiler translates hyphens in the COBOL word to underscore characters.

- external-name names a constant whose value is unknown at compile time but known at link time.

- Although the VALUE IS clause is not valid in the LINKAGE SECTION, the compiler allows such a specification, as a VSI extension. The clause, when specified in the LINKAGE SECTION, has no effect on program execution, and is flagged with an informational VSI extension diagnostic.

**Additional References**

- PROGRAM-ID paragraph in Chapter 3, Identification Division
- PICTURE clause
- USAGE clause
- Section 1.2.1.1, “User-Defined Words” in Section 1.2.1, “COBOL Words”
- Section 5.2.2: COBOL Standard Alignment Rules

**Examples**

1. The following is an example of initializing alphanumeric data items:

   01  ITEMA  PIC X(20) VALUE IS "12345678901234567890".
   01  ITEMB  PIC XX VALUE IS "NH".

2. The following is an example of initializing numeric data items:

   01  ITEMX  PIC S9999 VALUE IS -39.
   01  ITEMZ  PIC 9 VALUE ZERO.

3. The following is an example of assigning condition-name values:

   01  ITEMC  PIC 99.
   88  VAL1 VALUE IS 4.
   88  VAL2 VALUE IS 5 THRU 9 12.
   88  VAL3 VALUES ARE 10 14 THRU 23 27 29 30.
   88  VAL4 VALUES ARE 0 THRU 49, 51 THRU 99.
   88  VAL5 VALUES ARE 0 10 20 30 40 50.

4. The VALUE IS EXTERNAL clause allows a COBOL program to equate a mnemonic system constant to a value representing a return status code rather than the numeric equivalent. The following are some examples of this clause:

   On OpenVMS

   WORKING-STORAGE SECTION.
* System Services

01 BADHEADER PIC S9(9) COMP VALUE IS EXTERNAL SS$_BADFILHDR.

01 BADNAME PIC S9(9) COMP VALUE IS EXTERNAL SS$_BADFILENAME.

01 NORMAL PIC S9(9) COMP VALUE IS EXTERNAL SS$_NORMAL.

* Record Management Services

01 RMSDEV PIC S9(9) COMP VALUE IS EXTERNAL RMS$_DEV.

* Database

01 DBMDBGUSY PIC S9(9) COMP VALUE IS EXTERNAL DBM$_DBBUSY.

01 DBMEND PIC S9(9) COMP VALUE IS EXTERNAL DBM$_END.

* Run-Time Library

01 LIBINVARG PIC S9(9) COMP VALUE IS EXTERNAL LIB$_INVARG.

01 LIBINVSCRPOS PIC S9(9) COMP VALUE IS EXTERNAL LIB$_INVSCRPOS.

PROCEDURE DIVISION.

OPEN...
IF RMS-STS = BADHEADER PERFORM...
IF RMS-STS = BADNAME PERFORM 100-FIX-NAME.

5. The following example shows the VALUE IS REFERENCE clause:

DATA DIVISION.
WORKING-STORAGE SECTION.
01 ITEM-LIST.
  02 ITEM-PROCESS-NAME.
    03 PIC S9(4) COMP VALUE 15.
    03 PIC S9(4) COMP VALUE EXTERNAL JPI$_PRCNAM.
    03 POINTER VALUE REFERENCE PROCESS-NAME.
    03 POINTER VALUE REFERENCE PROCESS-NAME-LENGTH.
  02 ITEM-USER-NAME.
    03 PIC S9(4) COMP VALUE 12.
    03 PIC S9(4) COMP VALUE EXTERNAL JPI$_USERNAME.
    03 POINTER VALUE REFERENCE USER-NAME.
    03 PIC S9(4) COMP VALUE 0.
  02 ITEM-CPU-TIME.
    03 PIC S9(4) COMP VALUE 4.
    03 PIC S9(4) COMP VALUE EXTERNAL JPI$_CPUTIM.
    03 POINTER VALUE REFERENCE CPU-TIME.
    03 PIC S9(4) COMP VALUE 0.
  02 ITEM-TURMINAL.
    03 PIC S9(4) COMP VALUE 7.
    03 PIC S9(4) COMP VALUE EXTERNAL JPI$_TERMINAL.
VALUE OF ID

Function

The VALUE OF ID clause specifies, replaces, or completes a file specification.

General Format

```
VALUE OF { ID file-name } is { file-name data-name }
```

[file-name]

is a nonnumeric literal. It contains the full or partial file specification.

[data-name]

is the data-name of an alphanumeric Working-Storage Section data item. It contains the full or partial file specification.

General Rules

1. Each file specification field in file-name augments the specification in the ASSIGN clause of the SELECT statement.

2. A file specification field in the VALUE OF ID clause overrides the corresponding field in the SELECT statement. If a file specification field is either in the SELECT statement or in the VALUE OF ID clause (but not in both), it becomes part of the file specification.

3. On Tru64 UNIX systems, if you specify a VALUE OF ID clause with which you specified an OpenVMS logical, you must use an environment variable, as follows:

```
VALUE OF ID "DISK1"
```

Define the environment variable using one of the following:

```
% setenv DISK1
% setenv DISK1 /usr/data/
% setenv DISK1 /usr/data/test1.dat
```

4. The number of bytes in the string making up file-name or data-name must not exceed 255.

Technical Notes
• If the associated file connector is an external file connector, all file description entries in the run unit that are associated with that file connector must define the same file specification. For a data-name it must be external and reference the same data item in all programs defining the file.

Additional References

• ASSIGN

• VSI COBOL User Manual, on exception condition handling

• On OpenVMS, OpenVMS Record Management Services Reference Manual in the OpenVMS documentation set
Chapter 6. Procedure Division

This chapter includes the general formats for all Procedure Division statements, describes their basic elements, and explains how to use them.

6.1. Verbs, Statements, and Sentences

A COBOL verb is a reserved word that expresses an action to be taken by the compiler or the object program. A verb and its operands make up a COBOL statement. One or more statements terminated by a separator period form a COBOL sentence.

At the statement level, actions can be further differentiated: actions taken by the object program can be conditional or unconditional. In some cases, the verb in the statement defines whether the action is conditional or unconditional. One verb, IF, always defines a conditional action. Other verbs, such as READ, can define conditional action when you use phrases with them that make the action conditional. PERFORM and MOVE are examples of verbs that always define unconditional action. Most often, however, whether an action is conditional or unconditional depends on not only which verb, but also which phrases you use in the statement.

There are four types of COBOL statements:

- **Compiler-directing statements** specify an action taken by the compiler during compilation. See Section 6.1.1, “Compiler-Directing Statements and Sentences” for more information.
- **Imperative statements** specify an unconditional action taken by the object program at run time. See Section 6.1.2, “Imperative Statements and Sentences” for more information.
- **Conditional statements** specify a conditional action taken by the object program at run time. See Section 6.1.3, “Conditional Statements and Sentences” for more information.
- **Delimited-scope statements** specify their explicit scope terminator. See Section 6.1.4, “Scope of Statements” for more information.

Table 6.1, “Types and Categories of COBOL Statements” shows the four types of COBOL statements. It also shows that the imperative statements are further subdivided into nine categories and specifies the verbs that each category includes. When associated phrases are not specified, the verb alone defines the category. For compiler-directing and conditional statements, type and category are synonymous.

Table 6.1. Types and Categories of COBOL Statements

<table>
<thead>
<tr>
<th>Type</th>
<th>Category</th>
<th>Verb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler-Directing</td>
<td>Compiler-Directing</td>
<td>COPY, REPLACE, USE, RECORD</td>
</tr>
<tr>
<td>Conditional</td>
<td>Conditional</td>
<td>ACCEPT ([NOT] AT END or ...)</td>
</tr>
</tbody>
</table>

Legend:

(1) Without the optional [NOT] ON SIZE ERROR phrase
(2) Without the optional [NOT] ON EXCEPTION or [NOT] ON OVERFLOW phrase
(3) Without the optional [NOT] INVALID KEY phrase
(4) Without the optional [NOT] AT END or [NOT] INVALID KEY phrase
(5) Without the optional [NOT] ON OVERFLOW phrase
(6) Without the optional [NOT] INVALID KEY or [NOT] END-OF-PAGE phrase
<table>
<thead>
<tr>
<th>Type</th>
<th>Category</th>
<th>Verb</th>
</tr>
</thead>
<tbody>
<tr>
<td>(NOT) ON EXCEPTION</td>
<td></td>
<td>ADD (NOT) ON SIZE ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CALL (NOT) ON EXCEPTION or (NOT) ON OVERFLOW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMPUTE (NOT) ON SIZE ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DELETE (NOT) INVALID KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DISPLAY (NOT) ON EXCEPTION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DIVIDE (NOT) ON SIZE ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EVALUATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MULTIPLY (NOT) ON SIZE ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ (NOT) AT END or (NOT) INVALID KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RETURN((NOT) AT END)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REWRITE (NOT) INVALID KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEARCH(AT END)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>START (NOT) INVALID KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STRING (NOT) ON OVERFLOW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBTRACT (NOT) ON SIZE ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNSTRING (NOT) ON OVERFLOW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRITE (NOT) INVALID KEY or (NOT) END-OF-PAGE</td>
</tr>
</tbody>
</table>

Legend:

(1) Without the optional [NOT] ON SIZE ERROR phrase
(2) Without the optional [NOT] ON EXCEPTION or [NOT] ON OVERFLOW phrase
(3) Without the optional [NOT] INVALID KEY phrase
(4) Without the optional [NOT] AT END or [NOT] INVALID KEY phrase
(5) Without the optional [NOT] ON OVERFLOW phrase
(6) Without the optional [NOT] INVALID KEY or [NOT] END-OF-PAGE phrase
<table>
<thead>
<tr>
<th>Type</th>
<th>Category</th>
<th>Verb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imperative</td>
<td>Arithmetic</td>
<td>ADD (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMPUTE (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DIVIDE (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSPECT (TALLYING)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MULTIPLY (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBTRACT (1)</td>
</tr>
<tr>
<td></td>
<td>Data-Movement</td>
<td>ACCEPT (DATE, DAY, DAY-OF-WEEK or TIME)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INITIALIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSPECT (REPLACING or CONVERTING)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MOVE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET (TO TRUE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STRING (5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNSTRING (5)</td>
</tr>
<tr>
<td></td>
<td>Ending</td>
<td>STOP</td>
</tr>
<tr>
<td>Imperative</td>
<td>Input-Output</td>
<td>ACCEPT (identifier or CONTROL KEY IN identifier)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLOSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DELETE (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DISPLAY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OPEN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ (4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REWRITE (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET (TO ON or TO OFF)</td>
</tr>
</tbody>
</table>

**Legend:**

1. Without the optional [NOT] ON SIZE ERROR phrase
2. Without the optional [NOT] ON EXCEPTION or [NOT] ON OVERFLOW phrase
3. Without the optional [NOT] INVALID KEY phrase
4. Without the optional [NOT] AT END or [NOT] INVALID KEY phrase
5. Without the optional [NOT] ON OVERFLOW phrase
6. Without the optional [NOT] INVALID KEY or [NOT] END-OF-PAGE phrase
<table>
<thead>
<tr>
<th>Type</th>
<th>Category</th>
<th>Verb</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>START (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STOP (literal)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNLOCK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRITE (6)</td>
</tr>
<tr>
<td></td>
<td>Inter-Program Communications</td>
<td>CALL (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CANCEL</td>
</tr>
<tr>
<td></td>
<td>Procedure-Branching</td>
<td>ALTER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CALL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONTINUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GO TO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PERFORM</td>
</tr>
<tr>
<td></td>
<td>Table-Handling</td>
<td>SEARCH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET (TO, UP BY, or DOWN BY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SORT</td>
</tr>
<tr>
<td></td>
<td>Ordering</td>
<td>MERGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RELEASE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RETURN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SORT</td>
</tr>
<tr>
<td></td>
<td>Report Writing</td>
<td>GENERATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INITIATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUPPRESS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TERMINATE</td>
</tr>
</tbody>
</table>

Legend:
(1) Without the optional [NOT] ON SIZE ERROR phrase
(2) Without the optional [NOT] ON EXCEPTION or [NOT] ON OVERFLOW phrase
(3) Without the optional [NOT] INVALID KEY phrase
(4) Without the optional [NOT] AT END or [NOT] INVALID KEY phrase
(5) Without the optional [NOT] ON OVERFLOW phrase
(6) Without the optional [NOT] INVALID KEY or [NOT] END-OF-PAGE phrase
Like statements, COBOL sentences also can be compiler-directing, imperative, or conditional. Sentence type depends upon the types of statements the sentence contains. Table 6.2, “Contents of COBOL Sentences” summarizes the contents of the three types of COBOL sentences. The remaining text in this section describes each type of statement and sentence in greater detail.

### Table 6.2. Contents of COBOL Sentences

<table>
<thead>
<tr>
<th>Type</th>
<th>Contents of Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imperative</td>
<td>One or more consecutive imperative statements ending with a period</td>
</tr>
<tr>
<td>Conditional</td>
<td>One or more conditional statements, optionally preceded by an imperative statement, optionally preceded by an imperative statement, terminated by the separator period</td>
</tr>
<tr>
<td>Compiler-Directing</td>
<td>Only one compiler-directing statement ending with a period</td>
</tr>
</tbody>
</table>
6.1.1. Compiler-Directing Statements and Sentences

A compiler-directing statement causes the compiler to take an action during compilation. The verbs COPY, REPLACE, RECORD, and USE define compiler-directing statements. A compiler-directing sentence can contain other statements but it must contain only one compiler-directing statement. The compiler-directing statement must be the last statement in the sentence and must be followed immediately by a period.

6.1.2. Imperative Statements and Sentences

An imperative statement specifies an unconditional action for the program. It must contain a verb and the verb's operands, and cannot contain any conditional phrases. For example, the following statements are imperative:

OPEN INPUT FILE-A

COMPUTE C = A + B

However, the following statement is not imperative because it contains the phrase, ON SIZE ERROR, which makes the program's action conditional:

COMPUTE C = A + B ON SIZE ERROR PERFORM NUM-TOO-BIG.

In the Procedure Division rules, an imperative statement can be a sequence of consecutive imperative statements. The sequence must end with: (1) a separator period or (2) any phrase associated with a statement that contains the imperative statement. For example, the following sentence contains a sequence of two imperative statements following the AT END phrase.

READ FILE-A AT END PERFORM NO-MORE-RECS

DISPLAY "No more records." END-READ.

An imperative sentence contains only imperative statements and ends with a separator period.

6.1.3. Conditional Statements and Sentences

A conditional statement determines a condition's truth value. (A truth value is either a yes or no answer to the question, “Is the condition true?”.) The statement uses the truth value generated by the program to determine subsequent program action.

Conditional statements are as follows:

• An EVALUATE, IF, RETURN, or SEARCH statement
• An ACCEPT statement with the [NOT] AT END or [NOT] ON EXCEPTION phrase
• A DISPLAY statement with the [NOT] ON EXCEPTION phrase
• A READ statement with the [NOT] AT END or [NOT] INVALID KEY phrase
• A WRITE statement with the [NOT] INVALID KEY or [NOT] END-OF-PAGE phrase
• A DELETE, REWRITE, or START statement with the [NOT] INVALID KEY phrase
• An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) with the [NOT] ON SIZE ERROR phrase
• A STRING or UNSTRING statement with the [NOT] ON OVERFLOW phrase
• A CALL statement with the [NOT] ON EXCEPTION or [NOT] ON OVERFLOW phrase

A conditional sentence is a conditional statement that ends with a separator period. It can include an optional preceding imperative statement. For example, the following sentence is conditional even though it contains the imperative statement, GO TO PROC-A:
READ FILEA AT END GO TO PROC-A.

The program interprets this sentence to mean “If not at the end of the file, read the next record; otherwise, go to PROC-A.”

6.1.4. Scope of Statements

Scope terminators delimit the scope of some Procedure Division statements.

The scope of statements contained (nested) in other statements can also terminate implicitly. When statements are contained in other statements, a separator period (that terminates the sentence) terminates all nested statements as well.

In the following example, the separator period terminates the IF, MOVE, and PERFORM statements:

```
IF ITEMA = ITEMB
    MOVE ITEMC TO ITEMB
    PERFORM PROCA.
```

In the following example, the ELSE phrase of the IF statement terminates the scope of the READ and the first MOVE statements:

```
IF ITEMA = ITEMB
    READ FILEA
    AT END MOVE ITEMC TO ITEMB
ELSE
    MOVE ITEMD TO ITEME.
```

A delimited-scope statement is a special category of statement used in structured programming. A delimited-scope statement is any statement that includes its explicit scope terminator. See Section 6.3.4, “Explicit and Implicit Scope Terminators” for a list of explicit scope terminators.

A delimited-scope statement can be nested in another delimited-scope statement with the same verb. Then, each explicit scope terminator terminates the statement that begins with the closest unpaired preceding occurrence of the verb.

In the following example, the END-IF after the ADD statements (line 8) terminates the IF on line 5. The END-IF after the SUBTRACT (line 10) terminates the IF on line 3. The scope of the first IF statement (line 1) is terminated by the separator period on line 11.

```
1. IF ITEMA = ITEMB
2. ***MULTIPLY ITEMH BY ITEMI
3. ***IF ITEMI > 18
4. *****MOVE ITEMC TO ITEMD
5. *****IF ITEMD > ITEME
6. *******ADD ITEME TO ITEMF
7. *******ADD ITEMG TO ITEMH
8. ******END-IF
9. ******SUBTRACT 6 FROM ITEMH
10.****END-IF
11.***PERFORM PROCA.
```
6.2. Uniqueness of Reference

Every user-defined name in a COBOL program names a resource. (See the section on User-Defined Words in Section 1.2.1, “COBOL Words.”) To use a resource, however, a statement in a COBOL program must contain a reference that uniquely identifies that resource. Qualification, reference modification, and subscripting or indexing allow unique and unambiguous references to that resource. Qualified procedure-names allow uniqueness of reference to procedures, and qualified condition-names allow uniqueness of reference to condition-names.

When you assign the same name in separate (contained) programs to two or more occurrences of a resource, certain conventions apply that limit the scope of names. Name scoping and scope of names are COBOL language terms that describe the methods for resolving references to user-defined words in a contained program environment. (See Section 6.2.6: Scope of Names.)

Some user-defined words can be made available to every program in the run unit. (See the EXTERNAL clause in Chapter 5, Data Division.) These words are called external data. Other user-defined words can be made available to programs contained within the program that defines that resource. (See the GLOBAL clause in Chapter 5, Data Division.) These words are called global data.

6.2.1. Qualification

A reference to a user-defined word is unique if one or more of the following conditions exists:

• No other name has the same spelling, including hyphenation.

• It is part of a REDEFINES clause. (The reference following the word REDEFINES is unique because of the placement of the REDEFINES clause.) See the Syntax Rules for the REDEFINES clause.

• Scoping rules make it unique. (See Section 6.2.6: Scope of Names.)

A nonunique name within a hierarchy of names can be used in more than one place in your program. Unless you are redefining it, you must refer to this nonunique name using one or more higher-level names in the hierarchy. These higher-level names are called qualifiers. Using them to achieve uniqueness of reference is called qualification.

To make your reference unique, you need not specify all available qualifiers for a name, only those necessary to eliminate ambiguity.

Consider the following two record descriptions:

```
01 REC1.
   05 ITEMA     PIC XX.
   05 ITEMB     PIC X(20).
01 REC2.
   05 GROUP1.
      10 ITEMA    PIC 9(5).
      10 ITEMB    PIC X(3).
   05 GROUP2.
      10 ITEMC    PIC X(4).
      10 ITEDM    PIC X(8).
```

ITEMA and ITEMB appear in both record descriptions. Therefore, you must use qualifiers when you refer to these items in Procedure Division statements. For example, all of the following references to ITEMA are unique: ITEMA OF GROUP1, ITEMA OF REC1, ITEMA IN GROUP1 OF REC2.

Regardless of the preceding, you cannot use the same data-name as:

• The name of an external record and as the name of any other external data item in any program contained within or containing the program describing the external data record

• The name of an item possessing the global attribute and as the name of any other data item in the program describing the global data item
When a program is contained within a program or contains another program, specific conventions apply. (See Section 6.2.6, “Scope of Names”.)

The general formats for qualification are as follows:

**Format 1**

\[
\{ \text{data-name-1} \}
\{ \text{condition-name} \}
\left\{ \left\{ \left\{ \text{IN} \right\} \text{data-name-2} \right\} \cdots \left\{ \left\{ \text{IN} \right\} \text{file-name} \right\} \right\}
\{ \text{IN} \}
\{ \text{OF} \}
\{ \text{file-name} \}
\]

**Format 2**

\[
\text{paragraph-name} \left\{ \left\{ \left\{ \text{IN} \right\} \right\} \text{section-name} \right\} \]

**Format 3**

\[
\text{text-name} \left\{ \left\{ \left\{ \text{IN} \right\} \right\} \text{library-name} \right\} \]

**Format 4**

\[
\text{LINAGE-COUNTER} \left\{ \left\{ \left\{ \text{IN} \right\} \right\} \text{file-name} \right\}
\]

**Format 5**

\[
\left\{ \left\{ \left\{ \text{IN} \right\} \right\} \text{report-name} \right\}
\]

**Format 6**

\[
\left\{ \left\{ \left\{ \text{IN} \right\} \right\} \text{data-name-3} \left\{ \left\{ \left\{ \text{IN} \right\} \right\} \text{report-name} \right\} \right\}
\{ \text{IN} \}
\{ \text{OF} \}
\{ \text{report-name} \}
\]

**Format 7**

\[
\text{screen-name-1} \left\{ \left\{ \left\{ \text{IN} \right\} \right\} \text{screen-name-2} \right\} \cdots
\]

**Format 8**

\[
\{ \left\{ \left\{ \text{IN} \right\} \right\} \text{file-name} \}
\]

The following syntax rules apply to qualification:

1. Each reference to a nonunique, user-defined name must use a sequence of qualifiers that eliminates ambiguity from the reference.

2. A name can be qualified even if it does not need qualification. If more than one set of qualifiers ensures uniqueness, any set can be used.

3. IN and OF are equivalent.
4. In Format 1, each qualifier must be either the name associated with a level indicator, the name of a group to which the item being qualified is subordinate, or the name of a condition variable with which the condition-name being qualified is associated. (See Section 6.2.5, “Ensuring Unique Condition-Names”.) Qualifiers must be ordered from least- to most-inclusive levels in the hierarchy.

5. In Format 7, each qualifier must be the name of a group to which the item being qualified is subordinate. Qualifiers must be ordered from least- to most-inclusive levels in the hierarchy.

6. In Format 1, \textit{data-name-2} can be a record-name.

7. If the program contains explicit references to a \textit{paragraph-name}, the \textit{paragraph-name} cannot appear more than once in the same section. When a \textit{section-name} qualifies a \textit{paragraph-name}, the word SECTION cannot appear. A \textit{paragraph-name} need not be qualified in a reference from within the same section. You cannot reference a \textit{paragraph-name} or \textit{section-name} from any other program.

8. On OpenVMS, in Format 3, a COPY statement that accesses an OpenVMS Librarian \textit{library-record} must qualify \textit{text-name} with \textit{library-name}.

9. In Format 3, on Tru64 UNIX systems, the \textit{library-name} for the COPY statement will direct COPY to access the \textit{text-name} file from the \textit{library-name} subdirectory.

   See Chapter 8, \textit{Source Text Manipulation} for information on the COPY statement.

10. If the program has more than one file description entry with a LINAGE clause, every reference to LINAGE-COUNTER must be qualified.

11. If the program has more than one report description entry, every Procedure Division reference to LINE-COUNTER must be qualified.

12. In the Report Section, an unqualified reference to LINE-COUNTER is qualified implicitly by the name of the report in whose report description entry the reference is made. Whenever the LINE-COUNTER of a different report is referenced, LINE-COUNTER must be qualified explicitly by the report name associated with the different report.

13. If the program has more than one report description entry, every Procedure Division reference to PAGE-COUNTER must be qualified.

14. In the Report Section, an unqualified reference to PAGE-COUNTER is qualified implicitly by the name of the report in whose report description entry the reference is made. Whenever the PAGE-COUNTER of a different report is referenced, PAGE-COUNTER must be qualified explicitly by the report name associated with the different report.

15. On OpenVMS, if the program has more than one file description entry, every reference to RMS-STS, RMS-STV, and RMS-Filename must be qualified.

\textbf{6.2.2. Subscripts and Indexes}

Occurrences of a table are not individually named. You refer to them by using a subscript or index to specify their location relative to the table's beginning. Subscripting is a general operation; indexing is a special form of subscripting.

Unless otherwise specified by the rules for a statement, subscripts and indexes are evaluated once, at the beginning of a statement. If a statement contains rules describing the evaluation of subscripts, those rules also apply to the evaluation of indexes.

\textbf{Subscripting}
Subscripts can appear only in references to individual elements in a list, or table, of like elements that do not have individual data-names. (See the OCCURS clause in Chapter 5, Data Division.)

The general format for subscripting is as follows:

**Format 1**
```
{ data-name condition-name } { arithmetic-expression ... }
```

**Format 2**
```
argument ( ALL integer-1 data-name [-] integer-2 index-name [+] integer-3 ] ... )
```

All restrictions in the rules for subscripting also apply to indexing (See the following subsection describing the section called “Indexing”.) The following rules apply to subscripting:

1. A subscript can be represented by any arithmetic expression.

2. In Format 2, `argument` is an intrinsic function argument that is allowed to be repeated a variable number of times. Note that Format 1 also applies to intrinsic function arguments, but not with ALL subscripts. When ALL is specified as a subscript, the effect is as if each table element associated with that subscript position were specified. (For a list of the intrinsic functions that permit arguments with ALL subscripts, and for more information, see Chapter 7, Intrinsic Functions.) Also in Format 2, `data-name` is the data-name of a numeric integer elementary item.

3. Identifiers in subscript arithmetic expressions must refer to elementary numeric data items.

4. The lowest valid subscript value is 1. This value points to the first element of the table. Subscript values 2, 3, and so on, point to the next consecutive table elements.

5. The highest valid subscript value is the maximum number of occurrences of the item specified in the OCCURS clause.

6. The subscript or set of subscripts that identifies the table element is delimited by a balanced pair of left and right parentheses.

7. Each table element reference must include subscripting. However, the reference cannot include subscripting when it is one of the following:
   - The subject of a SEARCH statement
   - In a REDEFINES clause
   - In the KEY IS phrase of an OCCURS clause

8. The subscript or set of subscripts follows the table element's `data-name`. The `data-name` is then called a subscripted `data-name` or an identifier.

9. The number of subscripts following a table element reference must equal the number of dimensions in the table; that is, there must be a subscript for each OCCURS clause in the hierarchy that contains the table element and one for the table element itself.

10. A `data-name` can have up to 48 subscripts.

11. Subscripts must appear in the order of successively less inclusive dimensions of the table.

12. An arithmetic expression in a subscript cannot begin with a left parenthesis if the preceding arithmetic expression ends with a `data-name`. 
Note

Use the `check` compiler option with the `bounds` keyword for run-time upper- and lower-bound subscript range verification. The default action is not to check. For more information, refer to the COBOL online help file for your particular platform.

In the following examples, references to ITEME require two subscripts. The first subscript refers to the occurrence number of the most inclusive dimension, ITEMD (that contains ITEME).

Example 6.1. Subscripting Example

WORKING-STORAGE SECTION.
01 ITEMA PIC 99 COMP VALUE IS 3.
01 ITEMB PIC 99 COMP VALUE IS 5.
01 ITEMC VALUE IS "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
   03 ITEMD OCCURS 4 TIMES.
      05 ITEME OCCURS 6 TIMES PIC X.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEME (4,3)</td>
<td>##U</td>
</tr>
<tr>
<td>ITEME (ITEMA,ITEMB)</td>
<td>##Q</td>
</tr>
<tr>
<td>ITEME (ITEMA * 2 - 4, ITEMB - 2)</td>
<td>##I</td>
</tr>
<tr>
<td>ITEME (ITEMA * ITEMB / 15, (ITEMA + ITEMB) / #B 4)</td>
<td>##B</td>
</tr>
</tbody>
</table>

Indexing

Indexing is a special subscripting procedure. In indexing, you use the `INDEXED BY` phrase of the `OCCURS` clause to assign an `index-name` to each table level. You then refer to a table element using the `index-name` as a subscript.

The general format for indexing follows:

```plaintext
{ data-name \\
condition-name } { [ , index-name { [ "+" literal-2 ] } ] } ... 
```

All the restrictions in the rules for subscripting apply to indexing. (See the section called “Subscripting”.) The following rules apply only to indexing:

1. You must give `index-name` an initial value before using it. You can do this in:
   - A SET statement
   - A SEARCH statement with the ALL phrase
   - A PERFORM statement with the VARYING phrase

   Furthermore, only the statements in the previous list can change the value of `index-name`.

2. Indexing can be either direct or relative. Direct indexing means that the value of `index-name` or `literal-1` is the occurrence number. Relative indexing means that the occurrence number is the value of `index-name` plus or minus `literal-2`. `literal-2` must be an unsigned integer.
Note

Use the check compiler option with the bounds keyword for run-time upper- and lower-bound index range verification. The default is not to check. For more information, refer to the COBOL online help file for your particular platform.

Example 6.2, “Indexing Example” is similar to Example 6.1, “Subscripting Example” that illustrates subscripting. However, this example shows the use of index-names in references to the table, initializing indexes with the SET statement, and storing index-name values in index data items.

Example 6.2. Indexing Example

WORKING-STORAGE SECTION.
01 ITEMA USAGE IS INDEX.
01 ITEMB USAGE IS INDEX.
01 ITEMC VALUE IS "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
   03 ITEMD OCCURS 4 TIMES
       INDEXED BY DX.
   05 ITEME OCCURS 6 TIMES
       INDEXED BY EX PIC X.
PROCEDURE DIVISION.
PARA.
   SET DX TO 4.
   SET EX TO 1.
   DISPLAY ITEMD (DX).
   DISPLAY ITEME (DX, EX).
   DISPLAY ITEME (DX - 3, EX)
   SET ITEMA TO DX.
   SET ITEMB TO EX.

This example produces the following results:

1. :STUVWX
2. :S
3. :A

6.2.3. Reference Modification

Reference modification defines a subset of a data item by specifying its leftmost character and length.

General Format

{ data-name
   FUNCTION function-name ( [argument] . . . )
} {leftmost-character-position : [length] }

data-name must refer to a data item whose usage is DISPLAY.

function-name must refer to an alphanumeric function.

The specifications for leftmost-character-position and length must be arithmetic expressions.

Each character of a data item has an ordinal number corresponding to its position. The leftmost position is number 1; successive positions to the right are numbered 2, 3, 4, and so on. If the data-name's data description entry has a SIGN IS SEPARATE clause, the sign position is assigned an ordinal number in the data item.

For a data item defined as numeric, numeric edited, alphanumeric, alphabetic, or alphanumeric edited, reference modification operates as if the data item were redefined as an alphanumeric data item the same size as that referred to by data-name.
Unless otherwise specified by the rules for a statement, reference modification is evaluated only once, at
the beginning of a statement. Reference modification is evaluated immediately after subscripting or indexing
evaluation. Rules that describe the evaluation of subscripting for the various statements also apply to the evaluation
of reference modification.

The components of reference modification define the data item as follows:

- The evaluation of `leftmost-character-position` specifies the ordinal position of the data item's leftmost character.
  This position is relative to the leftmost character of the data item referred to by `data-name`. Evaluation of
  `leftmost-character-position` must result in an integer that is not less than 1, or greater than the number of
  characters in the data item referred to by `data-name`.

- The evaluation of `length` specifies the size of the unique data item. The evaluation must result in a positive
  integer. The sum of `leftmost-character-position` and `length` minus the value 1 must not exceed the number of
  characters in the data item referred to by `data-name`.

- If there is no `length`, the data item extends:
  - From and including the character identified by `leftmost-character-position` of the data item referred to by
    `data-name`
  - To and including the rightmost character of the data item referred to by `data-name`

The resulting unique data item is treated as an elementary item without the JUSTIFIED clause. It has the same
class and category as the data item referred to by `data-name`. However, the categories numeric, numeric edited,
and alphanumeric edited are considered category alphanumeric.

Reference modification is valid anywhere an alphanumeric identifier is allowed unless specific rules for a general
format prohibit it.

**Note**

Use the `check` compiler option with the `bounds` keyword for run-time upper- and lower-bound reference
modification range verification. The default is not to check. For more information, refer to the COBOL online
help file for your particular platform.

**Examples**

```cobol
WORKING-STORAGE SECTION.
01 ITEMA PIC X(15) VALUE IS "ABCDEFGHIJKLMNO".
01 ITEMB PIC 99 VALUE IS 10.
```

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEMA (2:3 )</td>
<td>BCD</td>
</tr>
<tr>
<td>ITEMA (ITEMB:2 )</td>
<td>JK</td>
</tr>
<tr>
<td>ITEMA (ITEMB / 2:ITEMB - 6 )</td>
<td>EFGH</td>
</tr>
<tr>
<td>ITEMA (ITEMB: )</td>
<td>JKLNO</td>
</tr>
</tbody>
</table>

**6.2.4. Identifiers**

In Procedure Division rules, the term `identifier` means the complete specification of a data item. The term refers
to all words required to make your reference to the item unique.

To reference a data item that is a function, a function-identifier is used. For information on functions, see Chapter 7,
*Intrinsic Functions*.

The general formats for identifiers are as follows:
For more information on the methods of uniquely specifying data items, see the following:

- Section 6.2.1: Qualification
- Section 6.2.2: Subscripts and Indexes
- Section 6.2.3: Reference Modification
- Section 6.2.6: Scope of Names

### 6.2.5. Ensuring Unique Condition-Names

If the name you use as a condition-name appears in more than one place in your program, it can be made unique through qualification, indexing, or subscripting. Your condition-name also is unique when the scope of names conventions by themselves ensure this as described in Section 6.2.6: Scope of Names.

The first qualifier for a condition-name can be the name of the item with which it is associated (the conditional variable). When qualifying condition-names, you must use the name of the conditional variable itself or the names of items that contain it.

References to a condition-name must have the same combination of subscripting or indexing that you use for the conditional variable.

The formats you use to ensure unique condition-names are the same as those used for an identifier, except condition-name replaces data-name.

In Procedure Division rules, the term condition-name refers to a condition-name along with any qualification and subscripting or indexing needed to avoid ambiguity.

### 6.2.6. Scope of Names

A contained COBOL program can refer to a user-defined word in its containing program if the user-defined word has the global attribute. (See Section 1.2.1.1, “User-Defined Words” in Section 1.2.1, “COBOL Words”.) Some user-defined words always have the global attribute, some never have the attribute (that is, they are local), and some might or might not, depending on the use of the GLOBAL clause. The following rules explain how to use different kinds of user-defined words and what kinds of local and global name scoping to expect.

1. The following types of user-defined words are always local and can be referenced only by statements and entries in the program declaring them:
   - Paragraph-name
   - Section-name
2. These user-defined words are always local when you define them in the Report Section. Only those statements and entries in the program containing the entries can reference them:
   - Condition-name
   - Data-name
• Record-name

3. The following user-defined word is always local when you define it in the Screen Section. Only those statements and entries in the program containing the entries can reference it.

• Screen-name

4. Because you cannot specify a Configuration Section for a program contained within another program, the following types of user-defined words are always global when declared in the Configuration Section. You can reference them only by statements and entries either in the program that contains the Configuration Section or in any program contained within that program.

• Alphabet-name
• Condition-name (declared in the Special Names paragraph)
• Mnemonic-name
• Symbolic-character-name
• Switch-name
• Class-name

5. The following user-defined words are global if you specify the GLOBAL clause:

• Condition-name (declared in the Data Division)
• Data-name
• File-name
• Index-name
• Record-name
• Report-name
• Segmented-key-name (if you specify the GLOBAL clause on the corresponding file-name)

Specific conventions for declarations and references apply to these types of user-defined words whenever the previous conditions do not apply.

Whenever duplicate names exist, a program always references the resource in its own program. If the resource is not in the referencing program, the following two conventions are used:

• Conventions for resolving program-name references
• Conventions for resolving other references

The next two sections describe these conventions.

6.2.6.1. Conventions for Resolving Program-Name References

The PROGRAM-ID paragraph of the Identification Division declares the program-name, a user-defined word to identify the program. Only the CALL and CANCEL statements and the END PROGRAM header can reference a program-name.

A run unit can contain multiple programs with duplicated program-names. However, when two programs have duplicate program-names, one of the two programs must directly or indirectly be contained within a separately compiled program that does not contain the program with the duplicated program-name.

The following rules regulate the scope of program-name:
1. Within a run unit, any separately compiled program can reference any other separately compiled program.

2. If a program-name does not have the COMMON attribute and it is contained directly within another program, the contained program can be referenced only by statements included in the directly containing program.

For example, in the run unit consisting of the three separately compiled programs illustrated in Example 6.3, “Separately Compiled Program 1”, Example 6.4, “Separately Compiled Program 2”, and Example 6.5, “Separately Compiled Program 3”:

- MAIN-PROGRAM (See in Example 6.3, “Separately Compiled Program 1”) directly contains program PROG-NAME-A and indirectly contains PROG-NAME-B, PROG-NAME-C, PROG-NAME-D, and PROG-NAME-F.

- PROG-NAME-B (See in Example 6.4, “Separately Compiled Program 2”)

- PROG-NAME-E (See in Example 6.5, “Separately Compiled Program 3”)


**Example 6.3. Separately Compiled Program 1**

```assembly
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN-PROGRAM.
.
.
.
CALL "PROG-NAME-B".
.
.
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-A.
.
.
.
CALL "PROG-NAME-B".
.
.
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-B.
.
.
.
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-C.
.
.
.
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-D.
.
.
```

213
CALL "PROG-NAME-B".

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-F.

CALL "PROG-NAME-B".

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-B.

Example 6.4. Separately Compiled Program 2

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-B.

Example 6.5. Separately Compiled Program 3

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-E.

CALL "PROG-NAME-B".

Example 6.6. Separately Compiled Program 1

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-D.

Example 6.7. Separately Compiled Program 2

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-A.

Example 6.8. Separately Compiled Program 3

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-F.


a. If a program-name has the COMMON attribute and it is contained directly within another program, the contained program can be referenced only by the following:

• Statements in the directly containing program

• Statements in any programs, directly or indirectly contained within the directly containing program, except statements in the program with the COMMON attribute and in any program it directly or indirectly contains

For example, in the run unit consisting of the three separately compiled programs illustrated in Example 6.6, “Separately Compiled Program 1”, Example 6.7, “Separately Compiled Program 2”, and Example 6.8, “Separately Compiled Program 3”:
• MAIN-PROGRAM (see  in Example 6.6, “Separately Compiled Program 1”) directly contains PROG-NAME-A  , and indirectly contains PROG-NAME-B (IS COMMON)  , PROG-NAME-C  , PROG-NAME-D  , PROG-NAME-F  , and PROG-NAME-G  .

• PROG-NAME-B (See  in Example 6.7, “Separately Compiled Program 2”.)

• PROG-NAME-E (See  in Example 6.8, “Separately Compiled Program 3”.)

The CALL “PROG-NAME-B” statement in PROG-NAME-A (See  in Example 6.6, “Separately Compiled Program 1”) references PROG-NAME-B IS COMMON because it is directly contained in PROG-NAME-A. The CALL “PROG-NAME-B” statement in PROG-NAME-F (See  in Example 6.6, “Separately Compiled Program 1”) references PROG-NAME-B IS COMMON because PROG-NAME-F is directly contained in PROG-NAME-A. The CALL “PROG-NAME-B” statement in PROG-NAME-G (See  in Example 6.6, “Separately Compiled Program 1”) references PROG-NAME-B IS COMMON because PROG-NAME-G is indirectly contained in PROG-NAME-A. The remaining CALL “PROG-NAME-B” statements ( and  in MAIN-PROGRAM and  in PROG-NAME-E) all reference the separately compiled program, PROG-NAME-B .

Example 6.6. Separately Compiled Program 1

IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN-PROGRAM.  .
.
.
   CALL "PROG-NAME-B".  
.
.
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-A.  
.
.
   CALL "PROG-NAME-B".  
.
.
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-B IS COMMON.  
.
.
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-C.  
.
.
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-D.  
.
.
   CALL "PROG-NAME-B".  
.
.
END PROGRAM PROG-NAME-D.
END PROGRAM PROG-NAME-C.
END PROGRAM PROG-NAME-B.
.
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-F. ①
.
.
CALL "PROG-NAME-B". ①
.
.
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-G. ①
.
.
CALL "PROG-NAME-B". ①
.
.
END PROGRAM PROG-NAME-G.
END PROGRAM PROG-NAME-F.
END PROGRAM PROG-NAME-A.
END PROGRAM MAIN-PROGRAM.

Example 6.7. Separately Compiled Program 2

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-B. ①
.
.
Example 6.8. Separately Compiled Program 3

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-E. ①
.
.
CALL "PROG-NAME-B". ②
.
.
6.2.6.2. Conventions for Resolving Other References

When a source program declares condition-names, data-names, file-names, record-names, report-names, and segmented-key-names, only the declaring source program can reference these names. The only exception is when names have the GLOBAL attribute and the program contains other programs.

For example, when a program such as PROG-NAME-A (See ① in Example 6.9, “Resolving References to Miscellaneous Names”) contains other programs (PROG-NAME-B ① and PROG-NAME-C ①), each program can define the same user-defined word. When such duplicated names are referenced, the rules for qualification of
names (see Section 6.2.1, “Qualification”) apply; and, if necessary, the following three hierarchical rules resolve any ambiguity:

1. References in a program to names defined in that program are resolved within the program. For example:
   - The following names: ①, ②, ③, and ④ are both defined and referenced within PROG-NAME-A.
   - The following names: ①, ②, ③, and ④ are both defined and referenced within PROG-NAME-B.
   - The following names: ⑤ and ⑥ are both defined and referenced within PROG-NAME-C.

2. A program cannot reference any condition-name, data-name, file-name, record-name, or report-name defined in a program it contains. For example, statements in PROG-NAME-A (See ①, ②, ③, and ④) cannot reference items in either PROG-NAME-B or PROG-NAME-C. Statements in PROG-NAME-B (see ⑤ through ⑥) cannot reference items in PROG-NAME-C.

3. If a program contains another program, any GLOBAL names defined in the containing program can be referenced by the following:
   - Statements in a directly contained program, provided that the directly containing program does not declare the same user-defined word, in which case, rule 1 applies. For example, compare the Procedure Division statement MOVE EXAMPLE1 … with MOVE EXAMPLE2 … in the same contained program.
   - Statements in an indirectly contained program, provided that neither the indirectly containing program nor any program in between declare the same name as a GLOBAL name. For example, compare the Procedure Division statement MOVE EXAMPLE3 … with MOVE EXAMPLE2 … in the same contained program.

Example 6.9. Resolving References to Miscellaneous Names

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-A. ①
DATA DIVISION.
FILE SECTION.
FD    FILE-NAME ... ②
  01    RECORD-NAME ... ③
FD    GLOBAL-FILE-NAME... IS GLOBAL...
WORKING-STORAGE SECTION.
  01    EXAMPLE1 ... IS GLOBAL...
  01    EXAMPLE2 ... IS GLOBAL...
  01    EXAMPLE3 ... IS GLOBAL...
  01    SWITCH-STATUS. ④
     88 ON   VALUE IS "1".
     88 OFF  VALUE IS "0".
  01    DATA-NAME ... ⑤
PROCEDURE DIVISION.
    MOVE DATA-NAME TO ... ⑥
    IF SWITCH-STATUS IS ON ... ⑦
    MOVE RECORD-NAME TO ...⑧
    OPEN INPUT FILE-NAME ...⑨

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-B. ⑩
DATA DIVISION.
FILE SECTION.
FD    FILE-NAME ...①①
  01    RECORD-NAME ...①②
WORKING-STORAGE SECTION.
  01    SWITCH-STATUS. ①③
88 ON VALUE IS "1".
88 OFF VALUE IS "0".
01 DATA-NAME...
01 EXAMPLE2...
01 EXAMPLE3... IS GLOBAL...
01 EXAMPLE4... IS GLOBAL...
PROCEDURE DIVISION.
  MOVE DATA-NAME TO ...
  IF SWITCH-STATUS IS ON ...
  MOVE RECORD-NAME TO ...
  OPEN INPUT FILE-NAME ...
  OPEN OUTPUT GLOBAL-FILE-NAME.
  MOVE EXAMPLE1 ...
  MOVE EXAMPLE2 ...
  MOVE EXAMPLE3 ...
  MOVE EXAMPLE4 ...
PROCEDURE DIVISION.
  OPEN OUTPUT GLOBAL-FILE-NAME.
  MOVE EXAMPLE1 ...
  MOVE EXAMPLE2 ...
  MOVE EXAMPLE3 ...
  MOVE EXAMPLE4 ...
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-C.
WORKING-STORAGE SECTION.
01 EXAMPLE2...
01 EXAMPLE4...
PROCEDURE DIVISION.
  OPEN OUTPUT GLOBAL-FILE-NAME.
  MOVE EXAMPLE1 ...
  MOVE EXAMPLE2 ...
  MOVE EXAMPLE3 ...
  MOVE EXAMPLE4 ...
END PROGRAM PROG-NAME-C.
END PROGRAM PROG-NAME-B.
END PROGRAM PROG-NAME-A.

If a data item possesses either or both the EXTERNAL or GLOBAL attributes and includes a table defining an index-name, that index-name also possesses either or both attributes.

If the file associated with a segmented key possesses either or both the EXTERNAL or GLOBAL attributes, that segmented key also possesses either or both attributes.

6.2.7. External and Internal Data

External data is associated with a run unit. Any program in the run unit describing the external data can reference that data. (See the EXTERNAL clause in Chapter 5, Data Division.) There is only one representation of an external data object.

Internal data is associated with a specific program.

External and internal data can have global names. (See the GLOBAL clause in Chapter 5, Data Division.)

6.3. Explicit and Implicit Specifications

The four types of explicit and implicit specifications follow:

• Procedure Division references
• Control transfers
• Attributes

• Scope terminators

6.3.1. Explicit and Implicit Procedure Division References

A source program can refer to data items explicitly or implicitly in Procedure Division statements.

An explicit reference occurs when the name of the item is in a Procedure Division statement or copied into the Procedure Division by a COPY statement.

An implicit reference occurs under the following conditions:

• When a Procedure Division statement refers to an item whose name does not appear in the statement.

• During PERFORM statement execution. The PERFORM statement's control mechanism can initialize, change, and evaluate an index or data item referred to in the VARYING, AFTER, and UNTIL phrases. These implicit references occur only if the PERFORM statement execution involves the data item.

6.3.2. Explicit and Implicit Control Transfers

The mechanism that controls program flow implicitly transfers control from one statement to another in the order in which the statements appear in the source program. The transfer occurs in this sequence unless an explicit control transfer overrides it, or there is no next executable statement.

A program can contain both explicit and implicit changes to the control transfer mechanism.

Implicit control transfer can also occur when normal program flow changes without executing a procedure-branching statement. For example:

• A paragraph can execute under the control of another COBOL statement (such as PERFORM, USE, SORT, and MERGE). If the paragraph is the last in the controlling statement's range, an implied control transfer occurs from the last statement in the paragraph to the controlling statement's control mechanism.

An implicit control transfer occurs between the control mechanism of a PERFORM statement that causes iterative execution and the first paragraph in its range. The transfer occurs for each iterative execution of the paragraph.

• When a SORT or MERGE statement executes, an implicit control transfer occurs to associated input or output procedures.

• When the execution of a statement causes the execution of a Declaratives Section, the control transfer is implicit. Another implicit control transfer occurs after execution of the Declaratives Section.

• If the Procedure Division does not have any Declaratives Sections, the program's first executable statement is the first executable statement in the Procedure Division. Otherwise, the program's first executable statement is the first executable statement after the declaratives part of the Procedure Division.

An explicit control transfer is a change to the implicit control transfer mechanism caused only by execution of either:

• A procedure-branching statement

• A conditional statement

The EXIT procedure-branching statement causes an explicit control transfer only when it has the PROGRAM phrase.
The Procedure Branching statement ALTER does not cause an explicit control transfer. However, it affects the explicit control transfer of the associated GO TO statement.

The term *next executable statement* refers to the next COBOL statement to which control transfers according to these rules and those associated with each language element.

There is no next executable statement when the program has no Procedure Division. This is also the case after:

- The last statement in a Declaratives Section, when the paragraph in which it appears is not executing under the control of another COBOL statement
- The last statement in a program, when the paragraph in which it appears is not executing under the control of another COBOL statement
- A STOP RUN or EXIT PROGRAM statement, when execution control transfers outside of the COBOL program containing the statement
- An END PROGRAM header

When there is no next executable statement and control does not transfer out of the program, program control flow is undefined. However, an EXIT PROGRAM statement implicitly executes when the program is under the control of a CALL statement.

### 6.3.3. Explicit and Implicit Attributes

An explicit attribute is an attribute the program explicitly specifies. If the program does not explicitly specify an attribute, the attribute assumes a default; it is then an implicit attribute.

For example, a program need not specify USAGE for a data item. If it does not, the data item's implicit usage is DISPLAY.

### 6.3.4. Explicit and Implicit Scope Terminators

Scope terminators delimit the scope of some Procedure Division statements as described in Section 6.1.4, “Scope of Statements”.

The following are explicit scope terminators:

<table>
<thead>
<tr>
<th>END-ACCEPT</th>
<th>END-ADD</th>
<th>END-CALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>END-COMPUTE</td>
<td>END-DELETE</td>
<td>END-DIVIDE</td>
</tr>
<tr>
<td>END-EVALUATE</td>
<td>END-IF</td>
<td>END-MULTIPLY</td>
</tr>
<tr>
<td>END-PERFORM</td>
<td>END-READ</td>
<td>END-RETURN</td>
</tr>
<tr>
<td>END-REWRITE</td>
<td>END-SEARCH</td>
<td>END-START</td>
</tr>
<tr>
<td>END-STRING</td>
<td>END-SUBTRACT</td>
<td>END-UNSTRING</td>
</tr>
<tr>
<td>END-WRITE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following are implicit scope terminators:

- At the end of a sentence the separator period terminates the scope of all previously unterminated statements.
- In a statement containing another statement the next phrase of the containing statement after the end of the contained statement terminates the scope of all unterminated contained statements. Examples are ELSE and WHEN.

### 6.4. Arithmetic Expressions

Whenever the term *arithmetic expression* appears in Procedure Division rules, it refers to one of the following:
An identifier of a numeric elementary item

A numeric literal

A figurative constant ZERO (ZEROS, ZEROES)

Two or more of the above separated by arithmetic operators

Two or more arithmetic expressions separated by arithmetic operators

A unary operator (a sign) can precede any arithmetic expression.

The identifiers and literals in an arithmetic expression must represent either of the following:

- Numeric elementary items

- Numeric literals on which arithmetic can be performed

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

### 6.4.1. Arithmetic Operators

Arithmetic expressions can use five binary and two unary arithmetic operators. A space must precede each operator and follow each binary operator.

The operators are as follows:

<table>
<thead>
<tr>
<th>Binary Arithmetic Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unary Arithmetic Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>The effect of multiplication by +1</td>
</tr>
<tr>
<td>-</td>
<td>The effect of multiplication by -1</td>
</tr>
</tbody>
</table>

### 6.4.2. Formation and Evaluation of Arithmetic Expressions

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

Parentheses can be used to specify the order in which elements in an arithmetic expression are evaluated. Expressions within parentheses are evaluated first. If you nest sets of parentheses, evaluation starts with the innermost set of parentheses and proceeds to the outermost set.

If the arithmetic expression contains no parentheses, the compiler evaluates arithmetic operators in the following hierarchical order:

<table>
<thead>
<tr>
<th>First</th>
<th>Unary plus and minus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>Third</td>
<td>Multiplication and division</td>
</tr>
</tbody>
</table>
This order also applies within a single set of parentheses.

If two or more operators are at the same hierarchical level, and parentheses do not specify the sequence of operations, evaluation proceeds from left to right.

Parentheses can eliminate ambiguities in logic when there are consecutive operations at the same hierarchical level, or change the normal hierarchical sequence of evaluation.

Consider the following expression:

\[(3 \times \text{ITEMA} - 2) / ((4 + \text{ITEMB}) \times -\text{ITEMA} - \text{ITEMC} ** 2)\]

The order of evaluation is as follows:
1. 4 + ITEMB
2. -ITEMA
3. 3 \times \text{ITEMA}
4. (The results of step 3) - 2
5. ITEMC ** 2
6. (The results of step 1) \times (the results of step 2)
7. (The results of step 6) - (the results of step 5)
8. (The results of step 4) / (the results of step 7)

Each left parenthesis in an arithmetic expression must have a matching right parenthesis, and each right parenthesis must have a matching left parenthesis.

If the first operator in an arithmetic expression is a unary operator, a left parenthesis ( () must immediately precede it when the arithmetic expression immediately follows an identifier or another arithmetic expression. For example:

```
CALL "OTHERPROG" USING ITEMA (-ITEMB) ITEMC.
```

The following rules apply to the evaluation of exponentiation:
1. If the value of an expression to be raised to a power is zero, the exponent value must be greater than zero. Otherwise, the size error condition exists. (See Section 6.6.4, "ON SIZE ERROR Phrase").
2. If the evaluation yields both a positive and negative real number, the positive number is the result.
3. If the evaluation yields no real number, the size error condition exists.

If the evaluation of the arithmetic expression results in an attempted division by zero, the size error condition exists.

When a statement with an arithmetic expression does not refer to a resultant identifier, the compiler stores the results of the arithmetic expression in an intermediate data item. (See Section 6.6.1, "Arithmetic Operations").

### 6.4.3. Standard Arithmetic (Alpha, I64)

When a floating-point data item is an operand in an arithmetic expression or an arithmetic statement, the rules for evaluation are described in Section 6.4.4.1, "FLOAT Arithmetic (Alpha, I64)".

When standard arithmetic is in effect, the following rules apply:
Procedure Division

1. Any operand of an arithmetic expression that is not already contained in a standard intermediate data item is
converted into a standard intermediate data item.
2. The size error condition is raised if the value is too large or too small to be contained in a standard intermediate
data item.
A standard intermediate data item is of the class numeric and the category numeric. It is the unique value zero or
an abstract, signed, normalized decimal floating-point temporary data item.
A standard intermediate data item has the unique value of zero or a value whose magnitude is in the range 10**-100
through 10**99 - 10**67, that is, (.100 000 000 000 000 000 000 000 000 000 00E-99) through (.999 999 999
999 999 999 999 999 999 999 99E+99) inclusive, with a precision of 32 decimal digits.
When the value of a standard intermediate data item is not zero, the fraction contains no digits to the left of the
decimal point and contains a digit other than zero to the immediate right of the decimal point.
A standard intermediate data item is rounded to 31 digits in the situations listed below.
1. When a standard intermediate data item is compared.
2. When a standard intermediate data argument is the argument of a function and there is no equivalent arithmetic
expression defined for the rules of the function, unless otherwise specified in the rules for a function or unless
situation 1, above, applies.
3. When a standard intermediate data item is being moved to a resultant-identifier for which the ROUNDED
phrase has not been specified. Rounding of a standard intermediate data item may cause the size error condition
to be raised.
When a standard intermediate data item is being moved to a resultant-identifier for which the ROUNDED phrase
is specified, the number of digits to which rounding occurs is as specified in the ROUNDED phrase.
When arithmetic expressions using addition, subtraction, multiplication, division, exponentiation, unary plus, and
unary minus are evaluated, the exact result is truncated to 32 significant digits, normalized, and stored in a standard
intermediate data item.

6.4.4. Native Arithmetic (Alpha, I64)
When a floating-point data item is an operand in an arithmetic expression or an arithmetic statement, the rules for
evaluation are those described in Section 6.4.4.1, “FLOAT Arithmetic (Alpha, I64)”.
When native arithmetic is in effect, the following rules apply:
1. If the result of an arithmetic expression can be represented without loss of significance in 38 decimal digits or
less, then decimal or computational operations are used to evaluate the expression.
2. When it is possible for an expression to produce more than 38 decimal digits, an intermediate data item is
selected based on the MATH_INTERMEDIATE qualifier.
The compiler assumes that all possible digit positions of a variable are significant.

6.4.4.1. FLOAT Arithmetic (Alpha, I64)
A double-precision binary floating-point intermediate data item is selected when /
MATH_INTERMEDIATE=FLOAT is specified. On OpenVMS Alpha and I64 this is a G_floating or T_floating
data item; on Tru64 UNIX, this is a T_floating data item. Refer to the Alpha Architecture Reference Manual for
more information on floating-point data types and operations.
A G_floating data item has a sign bit, an 11-bit binary exponent, and a normalized 53-bit fraction with the redundant
most significant fraction bit not represented. The magnitude of a G_floating data item is in the approximate range
0.56 * 10**-308 through 0.9 * 10**308. The precision of a G_floating data item is approximately one part in
2**52, typically 15 decimal digits.
223


A T_floating data item has a sign bit, an 11-bit binary exponent, and a 52-bit fraction. VSI COBOL generates code that uses the finite, normalized, floating-point range capabilities of T_floating. The magnitude of a T_floating data item is in the approximate range $2.2 \times 10^{-308}$ through $1.8 \times 10^{308}$. The precision of a T_floating data item is approximately one part in $2^{52}$, typically 15 decimal digits.

When the destination of an arithmetic statement is a floating-point data item, normal rounding takes place.

When an arithmetic expression references a floating-point operand, floating-point operations are used to evaluate the expression, and the result is represented in a floating-point intermediate data item. Floating-point operations use normal rounding; implicit conversions to integer are chopped. VSI COBOL provides support for finite (normalized) floating-point values only.

When arithmetic expressions using addition, subtraction, multiplication, division, exponentiation, unary plus, and unary minus are evaluated, the exact result is truncated to 53 significant bits, normalized, and stored in a floating-point intermediate data item.

### 6.4.4.2. CIT3 Arithmetic (Alpha, I64)

A decimal floating-point intermediate data item is selected when the qualifier `/MATH_INTERMEDIATE=CIT3` is specified.

A CIT3 intermediate data item has the unique value of zero or a value whose magnitude is in the range $10^{-100}$ through $10^{99} - 10^{81}$, that is, $(.100 \ 000 \ 000 \ 000 \ 000 \ 000E-99)$ through $(.999 \ 999 \ 999 \ 999 \ 999E+99)$ inclusive, with a precision of 18 decimal digits.

When a CIT3 intermediate data item is being moved to a resultant-identifier for which the ROUNDED phrase is specified, the number of digits to which rounding occurs is as specified in the ROUNDED phrase; when the ROUNDED phrase is not present, no rounding takes place.

When arithmetic expressions using addition, subtraction, multiplication, division, exponentiation, unary plus, and unary minus are evaluated, the exact result is truncated to 18 significant digits, normalized, and stored in a CIT3 intermediate data item.

### 6.4.4.3. CIT4 Arithmetic (Alpha, I64)

A decimal floating-point intermediate data item is selected when `/MATH_INTERMEDIATE=CIT4` is specified.

A CIT4 intermediate data item has the unique value of zero or a value whose magnitude is in the range $10^{-100}$ through $10^{99} - 10^{67}$, that is, $(.100 \ 000 \ 000 \ 000 \ 000 \ 000 \ 000E-99)$ through $(.999 \ 999 \ 999 \ 999 \ 999 \ 999 \ 999 \ 999 \ 999 \ 999E+99)$ inclusive, with a precision of 32 decimal digits.

Rounding rules for CIT4 arithmetic are the same as those described in Section 6.4.3, “Standard Arithmetic (Alpha, I64)”.

When arithmetic expressions using addition, subtraction, multiplication, division, exponentiation, unary plus, and unary minus are evaluated, the exact result is truncated to 32 significant digits, normalized, and stored in a CIT4 intermediate data item.

### 6.5. Conditional Expressions

A conditional expression specifies a condition the program must evaluate to determine the path of program flow. If the condition is true, the program takes one path; if it is false, the program takes another path. The IF, EVALUATE, PERFORM UNTIL, PERFORM VARYING, and SEARCH statements use conditional expressions. Any statement that can contain another imperative statement can contain a conditional expression.

A conditional expression can be either a simple or a complex condition. The types of simple conditions are the relation, class, condition-name, switch-status, sign, and success/failure conditions. Complex conditions are formed

---

*The blanks are added for readability.*
by using logical operators (AND, OR, NOT) with simple conditions. You can enclose conditions within any number of paired parentheses. However, embedding conditions this way has no effect on whether they are considered simple or complex.

### 6.5.1. Relation Conditions

A **relation condition** states a relation between two operands. The program compares the operands to determine whether the stated relation is true or false. The first operand is called the condition's subject. The second operand is called its object. Either operand can be an identifier, a literal, or the value of an arithmetic expression. The set of words that specifies the type of comparison is called the relational operator.

The format for a relation condition is as follows:

```plaintext
<table>
<thead>
<tr>
<th>Subject</th>
<th>Relational Operator</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ identifier-1</td>
<td>IS [ NOT ] GREATER THAN</td>
<td>{ identifier-2</td>
</tr>
<tr>
<td>literal-1</td>
<td>IS [ NOT ] &gt;</td>
<td>literal-2</td>
</tr>
<tr>
<td>arithmetic-expression-1</td>
<td>IS [ NOT ] &lt;</td>
<td>arithmetic-expression-2</td>
</tr>
<tr>
<td></td>
<td>IS [ NOT ] LESS THAN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IS [ NOT ] EQUAL TO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IS GREATER THAN OR EQUAL TO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IS [ NOT ] =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IS [ NOT ] &gt;=</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IS [ NOT ] &lt;=</td>
<td></td>
</tr>
</tbody>
</table>
```

You can compare two numeric operands regardless of their USAGE. However, if one or both of the operands are not numeric, they must have the same USAGE. If either operand is a group item, then the comparison is treated as nonnumeric, since group items are always considered alphanumeric.

You must refer to at least one variable in a relation condition; you cannot refer only to literals.

A space must precede and follow each word in the relational operator. However, NOT and the key word or relation character that follows NOT are treated as a unit.

The following relational operators are equivalent:

- **IS NOT GREATER THAN** is equivalent to **IS LESS THAN OR EQUAL TO**
- **IS NOT LESS THAN** is equivalent to **IS GREATER THAN OR EQUAL TO**

Table 6.3, “Relational Operators and Corresponding True Conditions” specifies valid true conditions that correspond to each relational operator.

### Table 6.3. Relational Operators and Corresponding True Conditions

<table>
<thead>
<tr>
<th>Relational Operator</th>
<th>True Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS GREATER THAN</td>
<td>Subject is greater than object.</td>
</tr>
<tr>
<td>IS &gt; THAN</td>
<td></td>
</tr>
<tr>
<td>IS NOT GREATER THAN</td>
<td>Subject is either less than or equal to object.</td>
</tr>
<tr>
<td>IS NOT &gt; THAN</td>
<td></td>
</tr>
</tbody>
</table>
The following two sections specify the rules that apply to numeric and nonnumeric comparisons in relation conditions.

### 6.5.1.1. Comparison of Numeric Operands

When both operands are numeric, their algebraic values are compared. The program performs the necessary conversion if the data descriptions of the operands specify different USAGE. When you use operands that are literals or arithmetic expressions, their length (in terms of the number of digits represented) is not significant.

Unsigned numeric operands are assumed to be positive for comparison. A zero value is always treated the same way, whether or not the operand contains a sign.

### 6.5.1.2. Comparison of Nonnumeric Operands

When one (or both) of the operands is nonnumeric, each operand is considered a string of alphanumeric characters. Therefore, the operands are compared according to the program's collating sequence. (See the OBJECT-COMPUTER paragraph in Chapter 4, Environment Division.)

If one of the operands is numeric, it must be either an integer literal or a data item described as an integer. The data item must be implicitly or explicitly described with USAGE DISPLAY. The treatment of the numeric data item is further affected by the following:

- If the nonnumeric operand is an elementary data item or a nonnumeric literal, the numeric data item is treated as though it were moved to an elementary alphanumeric data item of the same size. The content of this alphanumeric data item is then compared to the nonnumeric operand.
- If the nonnumeric operand is a group item, the numeric operand is treated as though it were moved to a group item of the same size. The content of this group item is then compared to the nonnumeric operand.
- When a numeric operand contains a sign, its sign is part of the string only if the other operand is a group item. Otherwise, the sign is removed and is not part of the comparison.

The two operands are compared character by character, beginning at the left end of each string. When the operation finds an unequal character pair, it uses that pair to evaluate the comparison. The greater operand is the one that contains the character with the higher collating sequence position. If the operands are of unequal size, the shorter operand is treated as if it were extended on the right with spaces to make it the same size as the other. Therefore, ABCD is greater than ABC (unless the program's collating sequence dictates otherwise).

<table>
<thead>
<tr>
<th>Relational Operator</th>
<th>True Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS LESS THAN</td>
<td>Subject is less than object.</td>
</tr>
<tr>
<td>IS &lt; THAN</td>
<td></td>
</tr>
<tr>
<td>IS NOT LESS THAN</td>
<td>Subject is either greater than or equal to object.</td>
</tr>
<tr>
<td>IS NOT &lt; THAN</td>
<td></td>
</tr>
<tr>
<td>IS EQUAL TO</td>
<td>Subject is equal to object.</td>
</tr>
<tr>
<td>IS = TO</td>
<td></td>
</tr>
<tr>
<td>IS NOT EQUAL TO</td>
<td>Subject is either greater than or less than object.</td>
</tr>
<tr>
<td>IS NOT = TO</td>
<td></td>
</tr>
<tr>
<td>IS GREATER THAN OR EQUAL TO</td>
<td>Subject is greater than or equal to object.</td>
</tr>
<tr>
<td>IS &gt;=</td>
<td></td>
</tr>
<tr>
<td>IS LESS THAN OR EQUAL TO</td>
<td>Subject is less than or equal to object.</td>
</tr>
<tr>
<td>IS &lt;=</td>
<td></td>
</tr>
</tbody>
</table>
Comparisons of Index-Names or Index Data Items

A program can compare the following:

- Two index-names
- One index-name and one literal or data item (other than an index data item)
- One index-name and one index data item
- Two index data items

6.5.2. Class Condition

The class condition tests whether the contents of an operand are numeric or alphabetic. It also determines if an alphabetic operand contains only uppercase characters, only lowercase characters, or if an operand is in conformance with class-name. The general format is as follows:

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

The identifier must reference a data item whose usage is explicitly or implicitly DISPLAY or COMP-3. If the identifier is a function-identifier, it must reference an alphanumeric function.

The following rules apply to the NUMERIC test:

1. The test is true when the operand contains only the characters 0 to 9 and the operational sign (subject to the next rule); otherwise, it is false.

2. The operand must contain an operational sign if its PICTURE clause specifies a sign. If the PICTURE clause does not specify a sign, the operand must not contain one. If the operand contains a sign that is not specified, or if a sign is specified and the operand does not contain one, the NUMERIC test is false.

3. You cannot use the test for an operand described as alphabetic or a group item containing signed elementary items.

The following rules apply to the ALPHABETIC test:

1. The test is true when the operand contains only the characters A to Z, a to z, and the space; otherwise, it is false.

2. You cannot use the ALPHABETIC test for an operand described as numeric.

The ALPHABETIC-LOWERCASE test is true when the operand contains only the characters a to z, and the space; otherwise, it is false.

The ALPHABETIC-UPPERCASE test is true when the operand contains only the characters A to Z, and the space; otherwise, it is false.

The class-name test is true when the operand consists entirely of the characters listed in the definition of class-name in the SPECIAL-NAMES paragraph. The class-name test must not be used with an item whose data description describes the item as numeric.
NOT and the key word following it are treated as a unit. For example, NOT NUMERIC is a test for determining that the operand is nonnumeric.

### 6.5.3. Condition-Name Condition

The *condition-name condition* determines if a data item contains a value assigned to one of that item's condition-names. The term *conditional variable* refers to the data item. *condition-name* refers to a level 88 entry associated with that item.

The general format for this condition is:

```
condition-name
```

The condition is true if one of the values corresponding to *condition-name* equals the value of the associated conditional variable. The data description for a variable can associate *condition-name* with one or more ranges of values. In this case, the condition tests to determine if the value of the variable falls in the specified range (end values included).

The following example illustrates testing *condition-names* associated with both one value and a range of values:

```
WORKING-STORAGE SECTION.
01 STUDENT-REC.
   05 YEAR-ID PIC 99.
   88 FRESHMAN VALUE IS 1.
   88 SOPHOMORE VALUE IS 2.
   88 JUNIOR VALUE IS 3.
   88 SENIOR VALUE IS 4.
   88 GRADUATE VALUE IS 5 THRU 10.
.
PROCEDURE DIVISION.
.
IF FRESHMAN ... IF SOPHOMORE ... IF JUNIOR ... IF SENIOR ... IF GRADUATE ... |
```

<table>
<thead>
<tr>
<th>Condition-Name</th>
<th>Test Is True When the Value of the Conditional Variable YEAR-ID Equals:</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRESHMAN</td>
<td>1</td>
</tr>
<tr>
<td>SOPHOMORE</td>
<td>2</td>
</tr>
<tr>
<td>JUNIOR</td>
<td>3</td>
</tr>
<tr>
<td>SENIOR</td>
<td>4</td>
</tr>
<tr>
<td>GRADUATE</td>
<td>5, 6, 7, 8, 9, or 10</td>
</tr>
</tbody>
</table>

When your program evaluates a conditional variable and its *condition-name*, the procedure is the same as the one used with the relation condition. (See Section 6.5.1, “Relation Conditions”).

### 6.5.4. Switch-Status Condition

The *switch-status condition* tests the on or off setting of an external logical program switch. Its general format is as follows:
You use the SWITCH clause of the SPECIAL-NAMES paragraph to associate condition-name with a logical switch setting. (See the SPECIAL-NAMES paragraph in Chapter 4, Environment Division.) The condition is true if the switch setting in effect during program execution is the same one assigned to condition-name.

**Note**

The translated value of the OpenVMS Alpha or I64 logical name COB$SWITCHES or the Tru64 UNIX environment variable COBOL SWITCHES specifies logical program switch settings. (Refer to the description of program switches in the VSI COBOL User Manual.)

### 6.5.5. Sign Condition

The sign condition determines if the algebraic value of an arithmetic expression is less than, greater than, or equal to zero.

Its general format is as follows:

```
expression IS \{ NOT \}
```

An operand is defined as:

- POSITIVE, if its value is greater than zero
- NEGATIVE, if its value is less than zero
- ZERO, if its value equals zero

expression must contain at least one reference to a variable.

NOT and the key word following it are treated as a unit. For example, NOT ZERO tests for a nonzero condition.

### 6.5.6. Success/Failure Condition

The success/failure condition tests the return status codes of COBOL and non-COBOL procedures for success or failure conditions.

**General Format**

```
status-code-id IS \{ SUCCESS FAILURE \}
```

[status-code-id] must be a COMP integer represented by PIC 9(1 to 9) COMP or PIC S9(1 to 9) COMP.

You can use the SET statement to initialize or alter the status of status-code-id.

The SUCCESS class condition is true if you specify status-code-id IS SUCCESS and status-code-id is in a SUCCESS state. Otherwise, the SUCCESS class condition is false.
The FAILURE class condition is true if you specify `status-code-id IS FAILURE` and `status-code-id` is in a FAILURE state. Otherwise, the FAILURE class condition is false.

`status-code-id` is in the SUCCESS state when the low-order bit of `status-code-id` is 1. It is in the FAILURE state when its low-order bit is 0.

**Examples**

1. On OpenVMS, calling a non-COBOL procedure:

```cobol
WORKING-STORAGE SECTION.
01 RMS-EOF PIC S9(9) COMP VALUE EXTERNAL RMS$_EOF.
01 RETURN-STATUS PIC S9(9) COMP.
PROCEDURE DIVISION.
A000-BEGIN.
  .
  .
  CALL "LIB$GET_SCREEN"
    USING BY DESCRIPTOR INPUT-TEXT, PROMPT,
    BY REFERENCE OUT-LEN,
    GIVING RETURN-STATUS.
    IF RETURN-STATUS = RMS-EOF PERFORM CTRL-Z-TRAP-Routine.
    IF RETURN-STATUS IS FAILURE PERFORM FAILURE-ROUTINE.
  .
  .
  2. Calling a COBOL procedure:

IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN-PROGRAM.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 RETURN-STATUS PIC S9(9) COMP.
PROCEDURE DIVISION.
  .
  .
  CALL "SUB" GIVING RETURN-STATUS.
  IF RETURN-STATUS IS FAILURE PERFORM FAILURE-ROUTINE.
  .
  .
IDENTIFICATION DIVISION.
PROGRAM-ID. SUB.
  .
  .
WORKING-STORAGE SECTION.
01 RETURN-STATUS PIC S9(9) COMP.
PROCEDURE DIVISION GIVING RETURN-STATUS.
  .
  .
  IF A = B
    SET RETURN-STATUS TO SUCCESS
  ELSE
    SET RETURN-STATUS TO FAILURE.
```

230
6.5.7. Complex Conditions

You form complex conditions by combining or negating other conditions. The conditions being combined or negated can be either simple or complex.

The logical operators AND and OR combine conditions. The logical operator NOT negates conditions. A space must precede and follow each logical operator in your program.

The truth value of a complex condition depends upon the following:

- The truth value of each condition it contains
- The effect of the logical operators

Table 6.4, “How Logical Operators Affect Evaluation of Conditions” shows the effect of each logical operator in complex conditions.

Table 6.4. How Logical Operators Affect Evaluation of Conditions

<table>
<thead>
<tr>
<th>Logical Operator</th>
<th>Meaning and Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Logical conjunction. The truth value is true if both connected conditions are true. It is false if one or both connected conditions are false.</td>
</tr>
<tr>
<td>OR</td>
<td>Logical inclusive OR. The truth value is true if one or both connected conditions are true. It is false if both conditions are false.</td>
</tr>
<tr>
<td>NOT</td>
<td>Logical negation or reversal of truth value. The truth value is true if the original condition is false. It is false if the original condition is true.</td>
</tr>
</tbody>
</table>

Negated Simple Conditions

The logical operator NOT negates a simple condition. The truth value of a negated simple condition is the opposite of the simple condition's truth value. Thus, the truth value of a negated simple condition is true only if the simple condition's truth value is false. It is false only if the simple condition's truth value is true.

The format for a negated simple condition is as follows:

Combined and Negated Combined Conditions

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

The general format is as follows:

```
condition { AND condition } { OR condition } ...
```
In the general format, *condition* can be one of the following:

- A simple condition
- A negated simple condition
- A combined condition
- A negated combined condition; that is, NOT followed by a combined condition enclosed in parentheses
- Valid combinations of the preceding conditions (see Table 6.5, “Combinations of Conditions, Logical Operators, and Parentheses”)

You can use matched pairs of parentheses in a combined condition. You do not need to write parentheses if the condition combines two or more conditions with the same logical operator (either AND or OR). In this case, the parentheses have no effect on the condition's evaluation. However, you might have to use parentheses if you use a mixture of AND, OR, and NOT logical operators. In this case, the parentheses can affect the condition's evaluation.

When the relevant parentheses are missing from a complex condition, the evaluation order of the logical operators determines the conditions to which the specified logical operators apply and implies the equivalent parentheses. The evaluation order is NOT, AND, OR. Thus, specifying:

```
a OR NOT b AND c
```

implies and is equivalent to specifying:

```
a OR ( (NOT b) AND c )
```

(See also Section 6.5.9, “Condition Evaluation Rules”.)

Table 6.5, “Combinations of Conditions, Logical Operators, and Parentheses” shows the permissible combinations of conditions, logical operators, and parentheses.

### Table 6.5. Combinations of Conditions, Logical Operators, and Parentheses

<table>
<thead>
<tr>
<th>Element</th>
<th>In a Conditional Expression</th>
<th>In a Left-to-Right Element Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Can Element Be First?</td>
<td>Can Element Be Last?</td>
</tr>
<tr>
<td>simple-condition</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>OR or AND</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>NOT</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>### (</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>### )</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

For example, Table 6.5, “Combinations of Conditions, Logical Operators, and Parentheses” shows whether or not the following element pairs can occur in your program:

<table>
<thead>
<tr>
<th>Element Pair</th>
<th>Permitted?</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR NOT</td>
<td>Yes</td>
</tr>
<tr>
<td>NOT OR</td>
<td>No</td>
</tr>
<tr>
<td>NOT (</td>
<td>Yes</td>
</tr>
</tbody>
</table>
6.5.8. Abbreviated Combined Relation Conditions

When you combine simple or negated simple conditions in a consecutive sequence, you can abbreviate any of the relation conditions except the first. You do this by either:

- Omitting the subject of the relation condition
- Omitting both the subject and the relational operator of the condition
- Ensuring that a relation condition in the consecutive sequence contains a subject (or subject and relational operator) that is common with the preceding relation condition
- Ensuring that there are no parentheses in the consecutive sequence

The general format for abbreviated combined relation conditions is as follows:

\[
\text{relation-condition} \left\{ \begin{array}{c}
\text{AND} \\
\text{OR} \\
\text{NOT} \\
\text{relational-operator} \\
\text{object} \\
\end{array} \right\} \ldots
\]

The evaluation of a sequence of combined relation conditions proceeds as if the last preceding subject appears in place of the omitted subject and the last preceding relational operator appears in place of the omitted relational operator. The result of these substitutions must form a valid condition. (See Table 6.5, “Combinations of Conditions, Logical Operators, and Parentheses”.)

When the word NOT appears in a sequence of abbreviated conditions, its treatment depends upon the word that follows it. NOT is considered part of the relational operator when immediately followed by: GREATER, >, LESS, <, EQUAL, or =. Otherwise, NOT is considered a logical operator that negates the relation condition.

Table 6.6, “Expanded Equivalents for Abbreviated Combined Relation Conditions” shows abbreviated combined (and negated combined) relation conditions and their expanded equivalents:

<table>
<thead>
<tr>
<th>Abbreviated Combined Relation Condition</th>
<th>Expanded Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>a &gt; b AND NOT &lt; c OR d</td>
<td>((a &gt; b) AND (a NOT &lt; c)) OR (a NOT &lt; d)</td>
</tr>
<tr>
<td>a NOT = b OR c</td>
<td>(a NOT = b) OR (a NOT = c)</td>
</tr>
<tr>
<td>NOT a = b OR c</td>
<td>(NOT (a = b)) OR (a = c)</td>
</tr>
<tr>
<td>NOT (a GREATER b OR &lt; c)</td>
<td>NOT ((a GREATER b) OR (a &lt; c))</td>
</tr>
<tr>
<td>a / b NOT = c AND NOT d</td>
<td>((a / b) NOT = c) AND (NOT ((a / b) NOT = d))</td>
</tr>
<tr>
<td>NOT (a NOT &gt; b AND c AND NOT d)</td>
<td>NOT ((((a NOT &gt; b) AND (a NOT &gt; c)) AND (NOT (a NOT &gt; d))))</td>
</tr>
</tbody>
</table>

6.5.9. Condition Evaluation Rules

Parentheses can specify the evaluation order in complex conditions. Conditions in parentheses are evaluated first. In nested parentheses, evaluation starts with the innermost set of parentheses. It proceeds to the outermost set.
Conditions are evaluated in a hierarchical order when there are no parentheses in a complex condition. This same order applies when all sets of parentheses are at the same level (none are nested). The hierarchy is shown in the following list:

1. Values for arithmetic expressions
2. Truth values for simple conditions, in this order:
   a. Relation
   b. Class
   c. Condition-name
   d. Switch-status
   e. Sign
   f. Success/failure
3. Truth values for negated simple conditions
4. Truth values for combined conditions, in this order:
   a. AND logical operators
   b. OR logical operators
5. Truth values for negated combined conditions

In the absence of parentheses, the order of evaluation of consecutive operations at the same hierarchical level is from left to right.

6.6. Common Rules and Options for Data Handling

This section describes the rules and options that apply when statements handle data. Data handling includes the following:

- Arithmetic operations
- Multiple receiving fields in arithmetic statements
- The ROUNDED phrase
- The ON SIZE ERROR phrase
- The CORRESPONDING phrase
- The ON EXCEPTION phrase
- Overlapping operands and incompatible data
- I/O status
- The INVALID KEY phrase
- The AT END phrase
- The FROM phrase
• The INTO phrase

### 6.6.1. Arithmetic Operations

The arithmetic statements begin with the verbs ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT. When an operand in these statements is a data item, its PICTURE must be numeric and specify no more than 31 digit positions on Alpha or I64. However, operands do not have to be the same size, nor must they have the same USAGE. Conversion and decimal point alignment occur throughout the calculation.

When you write an arithmetic statement, you specify one or more data items to receive the results of the operation. These data items are called resultant identifiers. However, the evaluation of each arithmetic statement can also use an intermediate data item. An intermediate data item is a compiler-supplied signed numeric data item that the program cannot access. It stores the results of intermediate steps in the arithmetic operation before moving the final value to the resultant identifiers.

When the final value of an arithmetic operation is moved to the resultant identifiers, it is transferred according to MOVE statement rules. Rounding and size error condition checking occur just before this final move. (See the MOVE statement, Section 6.6.4: ON SIZE ERROR Phrase, and Section 6.6.3: ROUNDED Phrase.)

### 6.6.2. Multiple Receiving Fields in Arithmetic Statements

An arithmetic statement can move its final result to more than one data item. In this case, the statement is said to have multiple receiving fields (or multiple results). The statement operates as if it had been written as a series of statements. The following example illustrates these steps. The first statement in the example is equivalent to the four that follow it. (Temp is an intermediate data item.)

```plaintext
ADD a, b, c TO c, d (c), e
ADD a, b, c GIVING temp
ADD temp TO c
ADD temp TO d (c)
ADD temp TO e
```

### 6.6.3. ROUNDED Phrase

The ROUNDED phrase allows you to specify rounding at the end of an arithmetic operation. The rounding operation adds 1 to the absolute value of the low-order digit of the resultant identifier if the absolute value of the next least significant (lower-valued) digit of the intermediate data item is greater than or equal to 5.

When the PICTURE string of the resultant identifier represents the low-order digit positions with the P character, rounding or truncation is relative to the rightmost integer position for which the compiler allocates storage. Therefore, when PIC 999PPP describes the item, the value 346711 is rounded to 347000.

If you do not use the ROUNDED phrase, any excess low-order digits in the arithmetic result are truncated when the result is moved to the resultant identifiers.

### 6.6.4. ON SIZE ERROR Phrase

The ON SIZE ERROR phrase allows you to specify an action for your program to take when a size error condition exists.

The NOT ON SIZE ERROR phrase allows you to specify an action for your program to take when a size error condition does not exist.

The format is as follows:
Procedure Division

| NOT | ON SIZE ERROR stmt |

Stmt is an imperative statement.

Size error checking occurs after decimal point alignment. Rounding occurs before size error checking. Also, truncation of rightmost digits occurs before size error checking.

A size error condition is caused by the following:

- Division by zero or invalid evaluation of exponentiation (see Section 6.4.2, “Formation and Evaluation of Arithmetic Expressions”). Both actions terminate the arithmetic operation.

- The absolute value of an arithmetic operation's result exceeds the value that is specified by the PICTURE clause of one or more of the resultant identifiers.

- Evaluation of an arithmetic expression would cause the new value to be outside the allowed range for the intermediate data item.

In the second case above, the size error condition affects the contents of only those resultant identifiers for which the size error exists.

When a size error condition occurs and the statement contains an ON SIZE ERROR phrase:

1. When standard arithmetic is in effect, the values of those resultant identifiers for which the size error exists are the same as before the operation began; when native arithmetic is in effect, those values are undefined.

2. The values of those resultant identifiers for which no size error exists are the same as they would have been if the size error condition had not occurred for any of the resultant identifiers.

3. The imperative statement in the ON SIZE ERROR phrase executes.

4. The NOT ON SIZE ERROR phrase, if specified, is ignored.

5. Control is transferred to the end of the arithmetic statement unless control has been transferred by executing the imperative statement of the ON SIZE ERROR phrase.

6. When a size error occurs in any arithmetic statement with multiple results, your program must analyze the results to determine where the size error occurred.

When a size error condition occurs and the statement does not contain an ON SIZE ERROR phrase:

1. The values of those resultant identifiers for which the size error exists are undefined.

2. The NOT ON SIZE ERROR phrase, if specified, is ignored.

3. Control is transferred to the end of the arithmetic statement.

When a size error condition does not occur:

1. The ON SIZE ERROR phrase, if specified, is ignored.

2. The imperative-statement in the NOT ON SIZE ERROR phrase, if specified, is executed.

3. Control is transferred to the end of the arithmetic statement unless control has been transferred by executing the imperative statement of the NOT ON SIZE ERROR phrase.

If you use the ADD or SUBTRACT statements with the CORRESPONDING phrase, any individual operation can cause a size error condition. In this instance, the imperative statement in the ON SIZE ERROR phrase executes after all the individual additions or subtractions are complete.
6.6.5. CORRESPONDING Phrase

The CORRESPONDING phrase allows you to specify group items as operands in order to use their corresponding subordinate items in an operation. See the ADD, SUBTRACT, and MOVE statements.

The following rules apply to the identifiers of operands in a statement containing the CORRESPONDING phrase:

1. All identifiers must refer to group items.
2. The data description entries of these identifiers can contain a REDEFINES or OCCURS clause.
3. Identifiers can be subordinate to a data description entry that has a REDEFINES or OCCURS clause.
4. You cannot specify identifiers with level-number 66, 77 or 88, or the USAGE IS INDEX clause.
5. Identifiers cannot be reference-modified.

The following rules describe the requirements for correspondence between data items subordinate to the identifiers. In these rules, identifier-1 refers to the sending group item and identifier-2 refers to the group in which results of the operation are stored.

1. Data items subordinate to both identifier-1 and identifier-2 must have the same data-name.
2. All possible qualifiers for a data item contained in identifier-1 (up to but not including identifier-1), must be identical to all possible qualifiers for the matching item in identifier-2 (up to but not including identifier-2).
3. In an ADD or SUBTRACT statement, the CORRESPONDING phrase affects only elementary numeric data items. Other data items do not take part in the operation.
4. In a MOVE statement, either the sending or receiving subordinate item can be a group item, but both cannot be. The classes of the data items in any corresponding pair can be different, but the resulting move must be legal according to the MOVE statement rules. (See the MOVE statement.)
5. The CORRESPONDING phrase disallows data items with the following:
   - Level-number 66
   - Level-number 88
   - A data description entry containing a REDEFINES, OCCURS, or USAGE IS INDEX clause
     A data item subordinate to one that is not eligible for correspondence is also disallowed.
6. FILLER data items and their subordinates are disallowed.
7. Neither identifier-1 nor identifier-2 can be reference modified.

6.6.6. ON EXCEPTION Phrase

The ON EXCEPTION phrase allows execution of an imperative statement when an exception (or error) condition occurs.

The NOT ON EXCEPTION phrase allows execution of an imperative statement when an exception condition (or any other error condition) does not occur.

The format is as follows:

```
| NOT | ON EXCEPTION statement
```
**Procedure Division**

Statement is an imperative statement.

The ON EXCEPTION phrase of the CALL statement prevents control transfer of the CALL and triggers the execution of the imperative statement related to the CALL.

The ON EXCEPTION phrase of the ACCEPT statement (Formats 3 and 4) allows you to handle data entry errors when data is accepted into a numeric data field using ACCEPT WITH CONVERSION. For additional information, see the ACCEPT statement.

The ON EXCEPTION phrase allows execution of an imperative statement when an ACCEPT statement (Format 5) terminates unsuccessfully. When there is an applicable CRT STATUS clause, unsuccessful termination is indicated by a value of ‘1’ or ‘9’ in the first character of the CRT STATUS data item (see the SPECIAL-NAMES section of Chapter 4, *Environment Division*).

When an exception condition occurs and the statement contains an ON EXCEPTION phrase:

1. The imperative statement associated with the ON EXCEPTION phrase is executed.
2. The NOT ON EXCEPTION phrase, if specified, is ignored.
3. Control is transferred to the end of the statement unless control has been transferred by executing the imperative statement of the ON EXCEPTION phrase.

When an exception condition occurs and the statement does not contain an ON EXCEPTION phrase:

1. The NOT ON EXCEPTION phrase, if specified, is ignored.
2. The program terminates abnormally.

When an exception condition does not occur:

1. The imperative statement associated with the NOT ON EXCEPTION phrase, if specified, is executed.
2. The ON EXCEPTION phrase, if specified, is ignored.
3. Control is transferred to the end of the statement unless control has been transferred by executing the imperative statement of the NOT ON EXCEPTION phrase.

### 6.6.7. Overlapping Operands and Incompatible Data

When statements refer to data items, two conditions can occur that can make program results unpredictable.

Undefined results occur when a sending and receiving item in an arithmetic statement or an INITIALIZE, INSPECT, MOVE, SET, STRING, or UNSTRING statement share a part of their storage areas.

Procedure Division references to a data item are undefined when a data item's contents are incompatible with the class of data defined by the item's PICTURE clause, or (if the item is a function) its function definition. Conditional statements containing the class condition allow you to do the following:

- Determine whether or not an item contains numeric or alphabetic data.
- Specify corrective action when it does not.

See Section 6.5.2, “Class Condition” for more information on class condition.

### 6.6.8. I-O Status

If a file description entry has a FILE STATUS clause, a value is placed in the two-character FILE STATUS data item during execution of the following I/O statements:
The two characters from the FILE STATUS data item combine to form a file status value. The first character (Status Key 1), which occupies the leftmost character position in the item, represents a specific class of I/O operation (0–success, 1–at end, 2–invalid key, 3–permanent error, 4–logic error, or 9–VSI defined). The second character (Status Key 2), which occupies the rightmost position, provides additional information about the result of an I/O operation. In combination, Status Key 1 and Status Key 2 indicate the status of an I/O operation. For example, if you are interested in duplicate keys, you check for File Status 02.

When Status Key 1 contains 1, the AT END phrase executes. When Status Key 1 contains 2, the INVALID KEY phrase executes. When Status Key 1 contains 3, 4, or 9 the Declarative USE procedure executes. Any applicable USE AFTER EXCEPTION procedure executes after the FILE STATUS value is set.

Figure 6.1, “Possible Combinations of Status Keys 1 and 2” shows the possible combinations of Status Keys 1 and 2. In the figure, X indicates a valid combination of keys.

**Figure 6.1. Possible Combinations of Status Keys 1 and 2**

<table>
<thead>
<tr>
<th>Status Key 2</th>
<th>No Further Information</th>
<th>Sequence Error</th>
<th>Duplicate Key</th>
<th>No Record Found</th>
<th>Boundary Violation</th>
<th>File Not Present</th>
<th>No Valid Next Record</th>
<th>File Not Found</th>
<th>Close Error</th>
<th>Open Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(0)</td>
<td>(1)</td>
<td>(2)</td>
<td>(3)</td>
<td>(4)</td>
<td>(5)</td>
<td>(6)</td>
<td>(7)</td>
<td>(8)</td>
<td>(9)</td>
</tr>
<tr>
<td>Successful Completion (6)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>At End (1)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invalid Key (2)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Permanent Error (3)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logic Error (4)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP- Defined (5)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Status Key 1**

Status Key 1 indicates one of the following conditions when an input-output operation ends:

0 Successful Completion. The input-output statement executed successfully.

1. At End. A sequential READ statement unsuccessfully executed because of the following:
   - The file has no next logical record.
• An optional file was not present.
• The program did not establish a valid next record.

2. Invalid Key. The input-output statement executed unsuccessfully because of one of the following conditions:
   • Sequence Error
   • Duplicate Key
   • No Record Found
   • Boundary Violation on a relative or indexed file
   • Optional File Not Present

3. Permanent Error. The input-output statement executed unsuccessfully because of a boundary error for a sequential file. This value can also indicate an input-output error, such as data check, parity error, or transmission error.

4. Logic Error. The input-output statement was unsuccessfully executed as a result of an improper sequence of input-output operations that were performed on the file or as a result of violating a limit set by the user.

9. VSI defined. The input-output statement executed unsuccessfully because of a condition defined by VSI.

**Status Key 2**

Status Key 2 further describes the result of the input-output operation as follows:

• If no further information about the input-output operation is available, Status Key 2 contains 0.
• When Status Key 1 contains 0 (indicating successful completion), Status Key 2 can contain a 2, 4, 5, or 7.

2. Applies to a REWRITE, WRITE, or READ statement.
   • For REWRITE and WRITE statements it means that the record just written created a duplicate key value for at least one alternate record key for which duplicates are allowed.
   • For READ statements it means that the record just read has duplicate key values for the current key of reference.

4. Applies to a READ statement. It means the size of the record read does not agree with the size defined in the program.

5. Applies to the OPEN statement. It means that the optional file was not present when the OPEN statement executed. If the open mode is I-O or EXTEND, the file has been created.

7. Applies to the CLOSE and OPEN statements. It means one of the following:
   • The program tried to execute a CLOSE REEL/UNIT, a CLOSE NO REWIND, or a CLOSE FOR REMOVAL statement for a file on a nonreel/unit medium.
   • The program tried to execute an OPEN NO REWIND statement for a file on a nonreel/unit medium.

• When Status Key 1 contains 1 (indicating an at end condition), Status Key 2 describes the condition's cause:
  0 Indicates that the file has no next logical record or it indicates that a file you specified as optional is not present.
  4. Means that the relative record number of the record read was too big for the relative key data item.
• When Status Key 1 contains 2 (indicating an invalid key condition), Status Key 2 describes the condition's cause as follows:

1. Indicates a sequence error for a sequential access indexed file. This means that the program changed the prime record key value between a successfully executed READ statement and the next REWRITE statement for the file. This value can also indicate that the program violated sort order sequence requirements for successive record key values. (See the WRITE statement.)

2. Indicates a duplicate key value. The program tried to write or rewrite a record that would have created a duplicate key in an indexed file. This value can also mean that the program tried to write a record that would have created a duplicate in a relative file.

3. Means that the program could not find a record. The program tried to access a record identified by a key, but the record does not exist in the file, or the file you specified as optional is not present.

4. Indicates a boundary violation. The program tried to write beyond the boundaries defined for the file by the I/O system OpenVMS Record Management Services (RMS) on OpenVMS Alpha and I64 systems, or the program attempted a sequential WRITE statement and the number of significant digits in the relative record number is larger than the size of the relative key data item.

• When Status Key 1 contains 3 (indicating a permanent error condition), Status Key 2 describes the condition causes as follows:

0 Indicates that no more information is available. This value results from any input-output error that cannot be described by any other combination of values in Status Keys 1 and 2. For example, "filename too long" is indicated this way.

4. Indicates a boundary violation on a sequential file. This means that the program tried to write to a disk that was full.

5. Indicates that the program tried to open a file that does not exist.

7. Indicates that the program tried to create a file on a device that is not appropriate for the OPEN statement mode.

8. Indicates that the program tried to open a file that is closed with a lock.

9. Indicates a conflict of file attributes. The attributes of the file that the program attempted to open do not match the attributes of the file described in the program. The attributes that are checked are BLOCK SIZE, ORGANIZATION, INDEX KEYS, and MAXIMUM RECORD SIZE. (Refer to the Tru64 UNIX reference page or COBOL online help for information on the effect of the relaxed key checking option.)

• When Status Key 1 contains 4 (indicating an error in the program's logic), Status Key 2 describes the condition's cause:

1. Indicates that the program tried to open a file that is already open.

2. Indicates that the program tried to close a file that either: (a) is already closed, or (b) has not been opened during the program's execution.

3. Indicates that the program tried to execute either: (a) a DELETE or REWRITE statement without first successfully executing a READ or START statement, or (b) an UNLOCK RECORD statement without first establishing a current record.

4. Indicates that the program attempted to REWRITE a record that is not the same size as the record being replaced.

6. Indicates that the program did not establish a valid next record.

The values 10 and 46 can occur for the same READ operation when a program is in an infinite loop. In this case, the FILE STATUS data item contains the following sequence of values:
00, 00, … , 00, 10, 46, … , 46

7. Indicates the program tried a READ or START operation on a file that: (a) has not been opened, or (b) has been opened in a mode that is incompatible with the operation.

8. Indicates the program attempted a WRITE operation on a file that: (a) has not been opened, or (b) has been opened in a mode that is incompatible with the operation.

9. Indicates the program tried a DELETE or REWRITE on a file that: (a) has not been opened, or (b) has been opened in a mode that is incompatible with the operation.

• When Status Key 1 contains 9 (indicating a VSI defined condition), Status Key 2 further describes the condition, as follows:

0 Means that the record your program is reading has been locked by another access stream. Because the record is available in the record area, the input operation is successful. This condition results from using the REGARDLESS option. Without the REGARDLESS option, the same scenario causes a Status Key 2 value of 2.

1. Indicates that a file is locked. The access stream tried to open a file that had been locked by another program.

2. Means that a record is locked. The program tried to access a record that had been locked by another access stream.

In this case, the record is not available in the record area, so the input operation is unsuccessful.

5. Means that the program tried to open a file when there was not enough space on the device.

8. Indicates that an unspecified error occurred when the program attempted to close a file.

Appendix C, File Status Values lists all the possible file status values that can appear in the FILE STATUS data item, along with the I-O status condition corresponding to each value.

6.6.9. AT END Phrase

The AT END phrase specifies the action your program takes when an at end condition occurs (when Status Key 1 contains 1).

The NOT AT END phrase specifies the action your program takes if an at end (or any other error condition) does not occur.

The format is as follows:

```
| NOT | AT END stment
```

`stment` is one or more imperative statements.

When a program detects the end of a file, the condition is called the *at end condition*. The at end condition might occur as a result of ACCEPT, READ, RETURN, or SEARCH statement execution. (For additional information, see the previously mentioned statements.)

When an at end condition occurs and the statement contains an AT END phrase:

1. The imperative statement associated with the AT END phrase, if specified, executes.

2. The NOT AT END phrase, if specified, is ignored.

3. Control is transferred to the end of the ACCEPT, READ, RETURN, or SEARCH statement unless control has been transferred by executing the imperative statement of the AT END phrase.
When an at end condition occurs and the statement does not contain an AT END phrase:

1. If the at end condition is associated with a READ statement, the applicable USE AFTER EXCEPTION procedure, if specified, executes.
2. If the at end condition is associated with an ACCEPT, RETURN, or SEARCH statement, any USE procedure associated with that file is not applicable.
3. The NOT AT END phrase, if specified, is ignored.

When an at end condition does not occur, and no other exception condition exists:

1. The AT END phrase, if specified, is ignored.
2. The imperative statement associated with the NOT AT END phrase, if specified, is executed. Otherwise, control is transferred to the end of the ACCEPT, READ, RETURN, or SEARCH statement.

When an at end condition does not occur, and another exception condition does exist:

- The applicable USE AFTER EXCEPTION procedure, if specified, executes and control is then transferred according to the rules of the USE statement.
- If there is no applicable USE AFTER EXCEPTION procedure, the imperative statement associated with the NOT AT END phrase, if specified, is executed, unless the exception condition causes the run unit to terminate abnormally.
- If there is neither a USE AFTER EXCEPTION procedure nor a NOT AT END phrase, then control is transferred to the end of the statement, unless the exception condition causes the run unit to terminate abnormally.

### 6.6.10. INVALID KEY Phrase

The INVALID KEY phrase specifies the action your program takes when an invalid key condition is detected (when Status Key 1 contains 2) for the file being processed.

The NOT INVALID KEY phrase specifies the action your program takes when an invalid key condition (or any other error condition) is not detected for the file being processed.

The format is as follows:

```
| NOT | INVALID KEY statement
```

`statement` is one or more imperative statements.

The invalid key condition occurs when the I/O system cannot complete a COBOL DELETE, READ, REWRITE, START, or WRITE statement because of one of the following conditions:

- Sequence error
- Duplicate key when the COBOL program did not specify this condition
- No record found
- Boundary violation on a relative or indexed file
- Optional file not present

(For more information on these conditions, refer to Section 6.6.8, “I-O Status”.) When the invalid key condition occurs, execution of the statement that produced the condition is unsuccessful, and the file is unaffected. (For additional information, see the previously mentioned statements.)

When the invalid key condition occurs:

1. A value that indicates the invalid key condition is placed in the FILE STATUS data item for the file.
2. If the statement that caused the condition has the INVALID KEY phrase:
   a. Any USE AFTER EXCEPTION procedure is not executed.
   b. The imperative statement associated with the INVALID KEY phrase executes.
   c. The NOT INVALID KEY phrase, if specified, is ignored.
   d. Control is transferred to the end of the I-O statement unless control has been transferred by executing the imperative statement of the INVALID KEY phrase.

3. If the statement that caused the condition does not have an INVALID KEY phrase:
   a. The NOT INVALID KEY phrase, if specified, is ignored.
   b. Control is transferred to the applicable USE AFTER EXCEPTION procedure for the file.

When an invalid key condition does not occur, and no other exception condition exists:

1. The INVALID KEY phrase, if specified, is ignored.

2. The imperative statement associated with the NOT INVALID KEY phrase, if specified, is executed. Otherwise, control is transferred to the end of the I/O statement.

When an invalid key condition does not occur, and another exception condition does exist:

• The applicable USE AFTER EXCEPTION procedure, if specified, executes and control is then transferred according to the rules of the USE statement.
• If there is no applicable USE AFTER EXCEPTION procedure, the imperative statement associated with the NOT INVALID KEY phrase, if specified, is executed, unless the exception condition causes the run unit to terminate abnormally.
• If there is neither a USE AFTER EXCEPTION procedure nor a NOT INVALID KEY phrase, then control is transferred to the end of the statement, unless the exception condition causes the run unit to terminate abnormally.

6.6.11. FROM Phrase

**Format 1**

```
record-name FROM identifier
```

*record-name* and *identifier* must not refer to the same storage area.

The result of executing a RELEASE, REWRITE, or WRITE statement with the FROM phrase is equivalent to: (1) executing the statement “MOVE *identifier* TO *record-name*” according to the rules of the MOVE statement without the CORRESPONDING phrase, followed by (2) executing the same RELEASE, REWRITE, or WRITE statement without the FROM phrase.

After statement execution ends, the data in the area referenced by *identifier* is available to the program. The data is not available in the area referenced by *record-name*, unless there is an applicable SAME clause. (See I-O-CONTROL, the REWRITE statement, and the WRITE statement.)

6.6.12. INTO Phrase

The INTO phrase implicitly moves a current record from the record storage area into an identifier.

The format is as follows:

```
file-name INTO identifier
```
A READ or RETURN statement can have the INTO phrase if either of the following conditions is true:

- Only one record description is subordinate to the file description entry.
- All record-names associated with file-name and the data item associated with identifier describe a group item or an elementary alphanumeric item.

Executing a READ or RETURN statement with the INTO phrase is equivalent to: (1) executing the same statement without the INTO phrase, then (2) moving the current record from the record area to the area specified by identifier. The move occurs according to the rules of the MOVE statement without the CORRESPONDING phrase. The move does not occur for an unsuccessful execution of the READ or RETURN statement.

Subscript or index evaluation occurs after the input operation and immediately before the move.

The record is available to the program in both the record area and the area associated with the identifier.

6.7. Segmentation

VSI COBOL programs execute in a virtual memory environment. Therefore, programs need not manage physical memory by overlaying Procedure Division code. VSI COBOL provides support for segmentation only for compatibility with existing applications developed on older hardware such as the PDP-11. You should not use segmentation in newly written COBOL programs since segmentation results in the generation of extra code which might impact performance.

Segmentation controls the assignment of Procedure Division sections to fixed or independent segments. The optional segment-number in the section header determines the type of segment.

```
General Format

  section-name SECTION [segment-number].
```

[section-name] names a Procedure Division section.

[segment-number]

must be an integer in the range 0 to 99. If there is no segment-number in a section header, the implied segment-number is 0.

segment-number classifies a segment into fixed segments or independent segments. Sections with segment-numbers from 0 to 49 are in fixed segments. Those with segment-numbers from 50 to 99 are in independent segments.

Sections in the Declaratives part of the Procedure Division must have segment-numbers less than 50.

A segment consists of all sections that have the same segment-number.

Both fixed and independent segments are in their initial state the first time entered. A fixed segment appears to reside in memory at all times and is, therefore, in its last used state each time it is entered.

The state of an independent segment depends on how and when it receives control. On subsequent control transfers, VSI COBOL resets the segment's ALTERed GO TO statements to their initial states whenever an independent segment is entered in one of the following ways:

1. Explicitly, by means of a GO TO statement with a target within the section.
2. Explicitly, by means of an out-of-line PERFORM statement in another segment whose range is within the section.
3. Implicitly, when a SORT or MERGE statement in another segment specifies an input or output procedure within the section.
4. Implicitly, by transfer of control between consecutive statements from a segment with a different segment-number.

6.8. General Formats and Rules for Statements

Function

The Procedure Division contains the routines that process the files and data described in the Environment and Data Divisions.

General Format

**Format 1**

```
PROCEDURE DIVISION [ USING { data-name } ... ] | GIVING identifier |

DECLARATIVES,

{ section-name SECTION [ segment-number ],
  declarative-sentence [ paragraph-name .
  [ sentence . . . ] . . . ]
  } . . .
END DECLARATIVES.

{ section-name SECTION [ segment-number ].

  [ paragraph-name . [ sentence . . . ] . . . ]

  } . . .

**Format 2**

PROCEDURE DIVISION [ USING { data-name } ... ] | GIVING identifier |

[ paragraph-name . [ sentence . . . ] . . . ] . . .
```

Syntax Rules

1. The Procedure Division follows the Data Division.

2. The Procedure Division must begin with the Procedure Division header.

3. The end of the Procedure Division is indicated by one of the following:
   - The Identification Division header of another source program
   - The END PROGRAM header
   - The physical position in the Procedure Division after which no further processing occurs

4. A procedure consists of either:
   - One or more successive sections
   - One or more successive paragraphs

5. If one paragraph is in a section, all paragraphs must be in sections.
6. A procedure-name refers to a paragraph or section in the source program. It is either paragraph-name (which can be qualified) or section-name.

7. A section consists of a section header followed by zero or more successive paragraphs. A section ends immediately before the next section or at the end of the Procedure Division. In the declaratives part of the Procedure Division, a section can also end at the key words END DECLARATIVES. See the section called “Declaratives” for more information on declaratives.

8. A paragraph consists of a paragraph-name followed by a separator period, and by zero or more successive sentences. A paragraph ends immediately before the next paragraph-name or section-name or at the end of the Procedure Division. In the declaratives part of the Procedure Division, a paragraph can also end at the key words END DECLARATIVE. See the section called “Declaratives” for more information on declaratives.

9. sentence contains one or more statements terminated by a separator period.

10. A statement is a syntactically valid combination of words and symbols that begins with a COBOL verb.

11. identifier is the word or words necessary to refer uniquely to a data item.

**Procedure Division Header**

1. The Procedure Division header identifies and begins the Procedure Division. It consists of the reserved words PROCEDURE DIVISION and optional USING and GIVING phrases followed by a separator period.

2. The USING phrase is required only if the program is invoked by a CALL statement with a USING phrase.

3. The Procedure Division header USING phrase identifies the names used in the program to refer to arguments from the calling program. In the calling program, the USING phrase of the CALL statement identifies the arguments. The data items in the two USING phrase lists correspond positionally.

4. Each data-name in the USING phrase must be defined in the Linkage Section with a level-01 or level-77 entry.

5. Each data-name cannot appear more than once in the USING phrase.

6. In the USING phrase, data-name cannot have the external attribute.

7. In the USING phrase, the data description for data-name cannot contain a REDEFINES clause. However, the data description can be the object of a REDEFINES clause.

8. The Procedure Division header GIVING phrase specifies a function result of the program. The identifier must refer to an elementary integer numeric data item with COMP, COMP-1, or COMP-2 usage and no scaling positions. The identifier cannot be subscripted, but it can be qualified.

**Procedure Division Body**

1. The Procedure Division body consists of all Procedure Division text following the Procedure Division header.

**General Rules**

1. References to USING phrase data-names operate according to data descriptions in the called program's Linkage Section, regardless of the descriptions in the calling program.

2. The called program can refer, in its Procedure Division, to a Linkage Section data item only if the data item satisfies one of these conditions:
   - It is in the Procedure Division header USING phrase.
   - It is subordinate to data-name that is in the Procedure Division header USING phrase.
   - Its definition includes a REDEFINES or RENAMES clause, the object of which is in the Procedure Division header USING phrase.
• It is subordinate to an item that satisfies the previous conditions.
• It is a condition-name or index-name associated with a data item that satisfies any of the previous conditions.

3. On Alpha and I64 systems, when a called program returns control to the calling program, the return value is made available to the calling program in the data item specified in its CALL statement GIVING phrase. The value is moved to that data item according to the rules for the MOVE statement. If the calling program does not specify a GIVING phrase, then the return value is made available in the calling program's RETURN-CODE special register. Note that the value in the called program's RETURN-CODE is not returned to the caller.

4. On Alpha and I64 systems, if no GIVING phrase is specified in the called program, the value in the RETURN-CODE special register is made available to the calling program in the data item specified in the calling program's CALL statement GIVING phrase. The value is moved according to the rules for the MOVE statement. If the calling program does not specify a CALL GIVING phrase, the value in the called program's RETURN-CODE special register is made available to the calling program in the calling program's RETURN-CODE special register.

Table 6.7, “Relation of GIVING Phrase to RETURN-CODE Special Register (Alpha, I64)” shows the relationship between the GIVING phrase and RETURN-CODE.

<table>
<thead>
<tr>
<th>Called program has CALL GIVING X</th>
<th>Called program has PROCEDURE DIVISION GIVING Y</th>
<th>Called program puts result in</th>
<th>Calling program gets result in</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>Y (also RETURN-CODE)</td>
<td>X (moved from Y)</td>
</tr>
<tr>
<td>YES</td>
<td>NO</td>
<td>RETURN-CODE</td>
<td>X (moved from called program's RETURN-CODE)</td>
</tr>
<tr>
<td>NO</td>
<td>YES</td>
<td>Y (also RETURN-CODE)</td>
<td>RETURN-CODE (moved from Y)</td>
</tr>
<tr>
<td>NO</td>
<td>NO</td>
<td>RETURN-CODE</td>
<td>RETURN-CODE (moved from called program's RETURN-CODE)</td>
</tr>
</tbody>
</table>

Technical Notes

1. On Alpha and I64 systems, because the reserved word RETURN-CODE is one of the X/Open reserved words, you cannot use the compilation flag -rsv noxopen (for Tru64 UNIX systems) or the corresponding qualifier /RESERVED_WORDS = NOXOPEN (for OpenVMS systems) if your program uses the RETURN-CODE special register.

2. On Alpha and I64 systems, VSI COBOL supports passing status to the operating system for RETURN-CODE and PROCEDURE DIVISION GIVING when EXIT PROGRAM or STOP RUN is executed.

Four of the data types supported by PROCEDURE DIVISION GIVING can be used to communicate status to the operating system. Following is a summary of what is supported for both RETURN CODE and PROCEDURE DIVISION GIVING:

```
RETURN-CODE (Alpha and I-64 only*)
EXIT PROGRAM /STA=V3 yes
EXIT PROGRAM /STA=85 yes
STOP RUN yes
PROCEDURE DIVISION GIVING
EXIT PROGRAM /STA=V3 yes
```
EXIT PROGRAM /STA=85 yes
STOP RUN yes*

Data Types
COMP-1, COMP-2 no
PIC 9(04) COMP no
PIC S9(04) COMP no
PIC 9(09) COMP yes
PIC S9(09) COMP yes
PIC 9(18) COMP yes
PIC S9(18) COMP yes
PIC 9(31) COMP no
PIC S9(31) COMP no

This support is subject to the limitations on status handling imposed by the operating system. If PIC S9(18) COMP or PIC 9(18) COMP is used, the high-order 32 bits are truncated before the status is passed on to the operating system.

To display the operating system status information, do the following:

[UNIX]       echo $status
[OpenVMS]    show symbol $status

Additional References

• CALL statement
• USE statement
• Section 6.7: Segmentation

Example

The following is an example of a Procedure Division header:

WORKING-STORAGE SECTION.
01      RETURN-RESULT PIC 9(8) COMP.
LINKAGE SECTION.
01      ARG1.
       03 ARG2       PIC X(6).
       03 ARG3       PIC S9(6) COMP.
01      ARG4          PIC X(4).
PROCEDURE DIVISION USING ARG1 ARG4 GIVING RETURN-RESULT.
       .
       .
       .
MOVE 17 TO RETURN-RESULT.
EXIT PROGRAM.

ACCEPT

ACCEPT

Function

The ACCEPT statement makes low-volume data available to the program. The VSI extensions to the ACCEPT statement (Formats 3, 4, and 5) are COBOL language additions that facilitate video forms design and data handling. The WITH CONVERSION phrase and some other options in Format 1 are also VSI extensions.
Format 6 retrieves the number of arguments on the program run command line, Format 7 reads those command line arguments into designated program variables, and Format 8 reads environment variables and logicals into designated program variables.

[dest-item] is the identifier of a data item into which data is accepted.

[input-source] is a mnemonic-name defined in the SPECIAL-NAMES paragraph of the Environment Division.

[stmt] is an imperative statement executed when the relevant condition (at end or on exception) occurs.

[stmt2] is an imperative statement executed when the relevant condition (not at end or not on exception) occurs.

[stmt3] is an imperative statement executed when an attempt is made to read beyond the last argument on the command line, or if the command line argument does not exist.

[stmt4] is an imperative statement executed if the name of a referenced environment variable or logical has not been set, or if the referenced environment variable or logical does not exist.

[stmt5] is an imperative statement executed if the exception condition does not exist.

[line-num] is a numeric literal that specifies a line position on the terminal screen. line-num must be a positive integer; it cannot be zero.

[line-id] is the identifier of a data item that provides a line position on the terminal screen. It must be a positive integer; it cannot be zero.

[plus-num] is a numeric literal that increments the current value for line or column position, or that increments the value of line-id or column-id. plus-num can be zero or a positive integer.

[column-num] is a numeric literal that specifies a column position on the terminal screen. column-num must be a positive integer; it cannot be zero.

[column-id] is the identifier of a data item that provides a column position on the terminal screen. It must be a positive integer; it cannot be zero.

[prot-size-lit] is a numeric literal that specifies the maximum length of the video screen field into which data can be typed. prot-size-lit must be a positive integer; it cannot be zero.
[prot-size-item]
is the identifier of a numeric integer data item that specifies the maximum length of the video screen field into which data can be typed. prot-size-item must be a positive integer; it cannot be zero.

[prot-fill-lit]
is a single character alphanumeric literal that is used to initialize each character position of a protected video screen field into which data can be typed.

[def-src-lit]
is a nonnumeric literal or a figurative constant. However, it cannot be the figurative constant ALL literal.

[def-src-item]
is the identifier of an alphanumeric data item.

[key-dest-item]
is the identifier of a data item that defines a control key. key-dest-item must specify an alphanumeric data item at least four characters in length.

[screen-name]
is the name of a screen item defined in the SCREEN SECTION of the program.

[arg-count]
is a mnemonic name associated with ARGUMENT-NUMBER in the SPECIAL-NAMES paragraph in the Environment Division. It represents the number of arguments present on the run command line.

[arg-value]
is a mnemonic name associated with ARGUMENT-VALUE in the SPECIAL-NAMES paragraph in the Environment Division. It contains the value of the argument on the run command line specified by the current argument position indicator.

[envlog-value]
is a mnemonic name associated with ENVIRONMENT-VALUE in the SPECIAL-NAMES paragraph in the Environment Division. It contains the value of a selected environment variable or system logical.

**Syntax Rules**

**Format 3**

1. You cannot specify a phrase more than once for any dest-item.

2. When you use the DEFAULT phrase and the PROTECTED phrase without the SIZE option, the size of def-src-item or def-src-lit must be less than or equal to the size of dest-item.

3. When you use the DEFAULT phrase and the PROTECTED phrase with the SIZE option, the size of def-src-item and def-src-lit must be less than or equal to prot-size-lit. If prot-size-item is specified and the specified size at run time is less than the length of def-src-item or def-src-lit, reprompting occurs.

4. The FILLER phrase cannot be used with the EDITING phrase. If both are present, the FILLER phrase is ignored.

**Format 4**

5. You cannot specify a phrase more than once for any key-dest-item.
**Format 6 (Alpha, I64)**

6. *dest-item* must reference a data item described as an unsigned integer.

**Formats 7 and 8 (Alpha, I64)**

7. *dest-item* must reference an alphanumeric data item.

**General Rules**

**Format 1**

1. The ACCEPT statement transfers data from *input-source*. The transferred data replaces the contents of *dest-item*.

2. The ACCEPT statement transfers a stream of characters with no editing or conversion, unless the WITH CONVERSION phrase is specified. Data transfer begins with the leftmost character position of *dest-item* and continues to the right.

3. If the data does not completely fill *dest-item*, remaining character positions are filled with spaces. If the data is too long for *dest-item*, it is truncated on the right.

4. The ACCEPT statement treats *dest-item* as alphanumeric, regardless of its class, unless the WITH CONVERSION phrase is specified.

5. If there is no FROM phrase, the ACCEPT statement transfers data from the default system input device.

**Format 2**

6. The ACCEPT statement transfers data to *dest-item* according to the MOVE statement rules.

7. DATE, DAY, DAY-OF-WEEK, and TIME are not actual data items. Therefore, the source program must not describe them.

8. DATE has three elements. From left to right, they are as follows:
   - Year of century (four digits if you specify YYYYMMDD, two digits if you do not)
   - Month of year (two digits)
   - Day of month (two digits)

   The ACCEPT statement operates as if DATE were described in the program as an eight-digit or six-digit, unsigned, elementary, numeric integer data item (PIC 9(8) or PIC 9(6)).

   For example, June 3, 1997 is expressed as 19970603 or 970603.

   - DAY has two elements. From left to right, they are as follows:
     - Year of century (four digits if you specify YYYYDDDD, two digits if you do not)
     - Day of year (three digits)

     The ACCEPT statement operates as if DAY were described in the program as a seven-digit or five-digit, unsigned, elementary, numeric integer data item (PIC 9(7) or (PIC 9(5)).

     For example, the fifteenth day of 1998 is expressed as 1998015 or 98015.5

---

5VSI COBOL also supports four-digit years using the CURRENT-DATE intrinsic function (see Chapter 7, *Intrinsic Functions.*) VSI recommends the use of four-digit years.
The YYYYMMDD and YYYYDDD options are VSI extensions.

9. DAY-OF-WEEK is a one-digit item that represents the day of the week.

The ACCEPT statement operates as if DAY-OF-WEEK were described in the program as a one-digit, unsigned, elementary numeric integer data item.

The values of DAY-OF-WEEK range from 1 (for Monday) to 7 (for Sunday).

10. TIME represents elapsed time after midnight, as shown on a 24-hour clock. It has four, two-digit elements. From left to right, they are as follows:

   • Hours
   • Minutes
   • Seconds
   • Hundredths of a second

The ACCEPT statement operates as if TIME were described in the program as an eight-digit, unsigned elementary numeric integer data item (PIC 9(8)).

The time 6:13 PM is expressed as 18130000. The minimum and maximum values of TIME are 00000000 and 23595999.

**Formats 3 and 4**

12. The ACCEPT statement transfers data from a video terminal. The data replaces the contents of dest-item (Format 3), or key-dest-item (Format 4). Format 3 can also update key-dest-item.

13. The presence of either the LINE NUMBER phrase or the COLUMN NUMBER phrase implies NO ADVANCING. The cursor remains on the character position immediately following the position of the last input character or in the position immediately following the rightmost position in the protected area. (For example, ACCEPT … PROTECTED SIZE 10 LINE 1 COLUMN 1, leaves the cursor at line 1, column 11, no matter what is typed in.) This is the default starting position of the next data item the program will input from or display upon the terminal.

14. If you do not use either the LINE NUMBER phrase or the COLUMN NUMBER phrase, data is accepted according to positioning rules for the Format 1 ACCEPT statement.

**Formats 3, 4, and 5**

15. The execution of certain extended ACCEPTs when the input source is assigned to a file (for example, in batch mode on OpenVMS systems), is a restriction. Syntax and actions that result in outputs from the ACCEPT operation (positioning, erasing, setting character attributes, reprompting, and protecting) to a nonvideo terminal are not supported and are ignored.

**LINE NUMBER Phrase (Formats 3 and 4)**

16. The LINE NUMBER phrase positions the cursor on a specific line of the video screen for data input.

17. If the LINE NUMBER phrase does not appear, but the COLUMN NUMBER phrase does, then data is accepted from the current line position and specified column position.

18. If line-num or the value of line-id is greater than the bottommost line position of the current screen, program results are undefined. (See Technical Notes.) Scrolling results if relative positioning is attempted past the bottom of the screen.
19. If you use `line-id` without its PLUS option, the line position is the value of `line-id`.

20. If you use `line-id` with its PLUS option, the line position is the sum of `plus-num` and the value of `line-id`.

21. If you use the PLUS option without `line-id`, the line position is the sum of `plus-num` and the value of the current line position.

22. If you use the PLUS option, but you do not specify `plus-num`, then PLUS 1 is implied.

23. Data input results are undefined if your program generates a value for `line-id` that is one of the following:
   - Zero
   - Negative
   - Greater than the bottommost line position of the current screen

**COLUMN NUMBER Phrase (Formats 3 and 4)**

24. The COLUMN NUMBER phrase positions the cursor on a specific column of the video screen.

25. If the COLUMN NUMBER phrase does not appear, but the LINE NUMBER phrase does, then data is accepted from column 1 of the specified line position.

26. If you use `column-id` without its PLUS option, the column position is the value of `column-id`.

27. If you use `column-id` with its PLUS option, the column position is the sum of `plus-num` and the value of `column-id`.

28. If you use the PLUS option without `column-id`, the column position is the sum of `plus-num` and the value of the current column position.

29. If you use the PLUS option, but do not specify `plus-num`, PLUS 1 is implied.

30. Data input results are undefined if the program generates a value for column position that is one of the following:
   - Zero
   - Negative
   - Greater than the last column position on the screen

**LINE NUMBER and COLUMN NUMBER Phrases (Format 5) (Alpha, I64)**

31. The LINE NUMBER and COLUMN NUMBER phrases together give the starting screen coordinates.

32. The position of each screen item within the referenced `screen-name` is offset from the LINE and COLUMN positions.

33. If either LINE or COLUMN is not specified, the default value is 1.

**ERASE Phrase (Formats 3 and 4)**

34. The ERASE phrase erases all, or part, of a line (or screen) before accepting data. You must specify SCREEN or LINE with the ERASE phrase.

35. If you use the TO END option, the ERASE phrase erases the line (or screen) from the implied, or stated, cursor position to the end of the line (or screen).
36. If you do not use the TO END option, the ERASE phrase erases the entire line (or screen).

**BELL Phrase (Formats 3 and 4)**

37. The BELL phrase rings the terminal bell before accepting data.

**CONTROL KEY Phrase (Formats 3 and 4)**

38. If you use the CONTROL KEY phrase, the characters representing PF keys and arrow keys, as well as TAB and RETURN, are legal terminator keys and can be accepted from the terminal. (See Technical Notes.)

39. `key-dest-item` stores the terminator key code; unused character positions, if any, are filled with spaces. (See Technical Notes.)

**ON EXCEPTION Phrase (Formats 3 and 4)**

40. The ON EXCEPTION phrase allows execution of an imperative statement when an exception (or error) condition occurs. ON EXCEPTION takes effect when illegal numeric data has been entered or there is an overflow on the left or right of the decimal point when CONVERSION is specified.

41. ON EXCEPTION can be used to detect numeric data entry errors only when accepting numeric data while CONVERSION is being used.

42. ON EXCEPTION can be used to detect end-of-file in any Format 3 or Format 4 ACCEPT statement.

43. ON EXCEPTION and AT END are mutually exclusive. If ON EXCEPTION is specified, the end-of-file indication is also a control key.

44. A DISPLAY statement within an ACCEPT ON EXCEPTION must be terminated (with, for example, END-DISPLAY) on Alpha and I64 systems. (If you are concerned with the different VAX behavior, refer to the appendix on compatibility in the VSI COBOL User Manual.)

**NOT ON EXCEPTION Phrase (Formats 3 and 4)**

45. The NOT ON EXCEPTION phrase allows execution of an imperative statement when an exception (or error) condition does not occur.

**ON EXCEPTION Phrase (Format 5, Alpha, I64)**

46. The ON EXCEPTION phrase allows execution of an imperative statement when the ACCEPT statement terminates unsuccessfully. When there is an applicable CRT STATUS clause, unsuccessful termination is indicated by a value of '1' or '9' in the first character of the CRT STATUS data item (see the SPECIAL-NAMES section of Chapter 4, Environment Division).

**NOT ON EXCEPTION Phrase (Format 5, Alpha, I64)**

47. The NOT ON EXCEPTION phrase allows execution of an imperative statement when the ACCEPT statement terminates successfully. When there is an applicable CRT STATUS clause, successful termination is indicated by a value of '0' in the first character of the CRT STATUS data item (see the SPECIAL-NAMES section of Chapter 4, Environment Division).

**NOT ON EXCEPTION Phrase (Formats 3 and 4; and Formats 5, 7, and 8, Alpha, I64)**

48. A DISPLAY statement within an ACCEPT [NOT] ON EXCEPTION statement must be terminated (with, for example, END-DISPLAY).
AT END Phrase (Formats 3 and 4)

49. The AT END phrase allows execution of an imperative statement when an end-of-file condition occurs.

50. AT END and ON EXCEPTION are mutually exclusive.

51. If AT END is specified, the end-of-file indication is also a control key. If you do not specify AT END or ON EXCEPTION, and end-of-file is entered, an error condition occurs.

NOT AT END Phrase (Formats 3 and 4)

52. The NOT AT END phrase allows execution of an imperative statement when an end-of-file condition (or other error) does not occur.

Format 3

UNDERLINED Phrase (Format 3)

53. The UNDERLINED phrase echoes input characters to the terminal with the underscore on character attribute.

54. When you use the UNDERLINED phrase with the PROTECTED phrase, the input field is underlined prior to accepting data.

BOLD Phrase (Format 3)

55. The BOLD phrase echoes input characters to the terminal with the bold on character attribute.

56. When you use the BOLD phrase with the PROTECTED phrase, the input field is visibly bolded prior to accepting data only if: (1) the underlined or reversed attributes are also specified, or (2) the video terminal screen is set to light background.

BLINKING Phrase (Format 3)

57. The BLINKING phrase echoes input characters to the terminal with the blink on character attribute.

58. When you use the BLINKING phrase with the PROTECTED phrase, the input field is visibly blinked prior to accepting data only if: (1) the underlined or reversed attributes are also specified, or (2) the video terminal screen is set to light background.

REVERSED Phrase (Format 3)

59. The REVERSED phrase echoes input characters to the terminal with the reverse video on character attribute.

60. When you use the REVERSED phrase with the PROTECTED phrase, the input field appears in reverse video prior to accepting data.

CONVERSION Phrase (Formats 1 and 3)

61. The CONVERSION phrase allows you to accept data into a field and achieve the same results as you would with the MOVE statement for non-floating-point items, and provides conversion from display, display scaled or E-notation to floating point in the case of floating-point data items. It enables validation of the accepted data and facilitates editing and alignment of data within dest-item. The effect of the CONVERSION phrase on data handling depends on the category of dest-item. (Numeric data can be described by any USAGE clause.)

62. When dest-item is numeric or numeric edited (other than floating point), the CONVERSION phrase:
• Converts input numeric data to a numeric literal (TRAILING SEPARATE SIGNED DISPLAY DECIMAL)
• Moves the result to dest-item (using MOVE statement rules)

63. When dest-item is floating point, the CONVERSION phrase:

• Converts input data to floating point (COMP-1 or COMP-2 as appropriate).
• Moves the converted result to the destination as if a numeric literal equivalent to the input data was moved to the destination with the MOVE statement.

64. When dest-item is numeric or numeric edited (other than floating point), and you use the CONVERSION phrase, valid input characters are as follows:

• 0 to 9
• Period (.), unless DECIMAL POINT IS COMMA is specified
• Comma (,), if DECIMAL POINT IS COMMA is specified
• Space (leading and trailing)
• Sign (+ or -)

The terminal operator can input space characters only as leading and trailing spaces. If this occurs, space characters are simply ignored during numeric conversion.

However, the operator cannot input space characters between numeric characters, between numeric characters and a decimal point, or between a sign and any other input character. When this occurs, the input data is invalid, and an error condition results.

The operator can input only one sign character and one decimal point character.

When the operator inputs a sign character, it must precede or follow all numeric characters and the decimal point.

The default sign character is a plus sign (+).

The default number of decimal places is zero.

65. When dest-item is floating point, and you use the CONVERSION phrase, valid input characters are as follows:

• <zero or more blanks>
• <“+”, “-”, or null>
• <zero or more decimal digits>
• <“.” or null> if DECIMAL POINT IS COMMA then <“,” or null>
• <zero or more decimal digits>

For example:

2.5E2
-0.08e4
10.0E-1
-2.14158E0

Note that numbers can be expressed in several ways. For example, the number 257.0 can be represented in any of the following ways:

257e0    2.57E2    0.000257E+6    2570E-1
66. When you use the CONVERSION phrase and dest-item is numeric, data input results in a conversion error condition if the operator enters any of the following:

- For fixed point numeric, too many characters on either side of the decimal point. (The PICTURE clause of dest-item determines this overflow condition.)

- For floating-point numeric, an exponent outside the valid range:
  - For IEEE double format (T_floating format) and G_floating format, the valid range is +308 to -308.
  - For IEEE single format (S_floating format), D_floating format, and F_floating format, the valid range is +38 to -38.
  - Invalid numeric data (for both fixed and floating-point numeric).

When one of these error conditions occurs, and you do not specify the ON EXCEPTION phrase: (1) the contents of dest-item do not change, (2) the terminal bell rings, (3) the input field is erased, and (4) the ACCEPT statement executes again. The input field is not erased if the EDITING phrase is used.

When one of these error conditions occurs, and you do specify the ON EXCEPTION phrase: (1) the contents of dest-item do not change, (2) the input field is left as if no error occurred, and (3) the imperative statement of the ON EXCEPTION phrase executes.

67. When dest-item is not numeric, the CONVERSION phrase moves input characters to dest-item as an alphanumeric string (MOVE statement rules apply). Therefore, data can be accepted into an alphanumeric edited field, or a reference modified field, and the JUSTIFIED clause, if it applies to dest-item, can take effect.

An overflow condition is not an error condition when dest-item is alphanumeric; in this case, right-end truncation occurs. However, you can specify the PROTECTED and SIZE phrases to limit the amount of input data when dest-item is alphanumeric.

68. When you use the CONVERSION phrase, and the operator types the terminator key prior to any data input:

- ZEROES (or spaces if BLANK WHEN ZERO is specified) are moved to a numeric or numeric edited dest-item, if you do not specify the DEFAULT phrase.

- SPACES are moved to an alphanumeric or alphanumeric edited dest-item, if you do not specify the DEFAULT phrase.

- However, the default value is moved to dest-item, if you do specify the DEFAULT phrase.

If the default value is not a valid value for dest-item, an error condition results. If ON EXCEPTION is used, an exception path is followed. If ON EXCEPTION is not used, an automatic reprompt for data occurs.

69. If you do not use the CONVERSION phrase, data is transferred to dest-item according to Format 1 ACCEPT statement rules when CONVERSION is not specified.

**PROTECTED Phrase (Format 3)**

70. The PROTECTED phrase limits the number of characters that can be entered from the terminal.

71. If you do not specify the PROTECTED phrase, the cursor remains on the character position immediately following the position of the last input character. This is the default starting position of the next data item you input from or display upon the terminal.

However, if you use the PROTECTED phrase to delimit the field of a data item, the cursor moves to the character position immediately following the last position of the input field. In this case, the default starting position of the next data item is always to the right of the input field, as determined by the SIZE phrase or PICTURE clause.

72. When you specify the PROTECTED phrase without the AUTOTERMINATE phrase:
• If the operator attempts to type beyond the rightmost position of the input field: (1) the terminal bell rings, (2) the cursor remains on the position to the right of the rightmost position, and (3) character entry attempts beyond the rightmost position are not echoed to the terminal screen.

• If the operator attempts to delete beyond the leftmost position of the input field: (1) the terminal bell rings, and (2) the cursor remains on the leftmost position.

73. If you use the PROTECTED phrase without the SIZE phrase, the maximum number of characters that the operator can enter is as follows:

• The number of characters in a fixed point `dest-item`

• Thirteen, if `dest-item` is COM-1 F_floating format or IEEE S_floating format

• Twenty two, if `dest-item` is COM-2 D_floating or G_floating format or IEEE T_floating format

For non-floating-point numeric items, the maximum number of characters allows for entry of sign and decimal point characters when these are implied by `dest-item`'s PICTURE clause. For example, if PIC S9(4)V99 is the PICTURE clause for `dest-item`, all of the following character strings are valid input:

```
2222.22
2222
.22
+2222.22
222.22-
-02.2
```

Although the sign does not represent a character position (and does not count toward item size), a space in the input field is allocated for it. A space is also allocated for the implied decimal point.

74. When you use the PROTECTED phrase without the NO BLANK or FILLER phrase, the input field is filled with spaces prior to accepting data. If you also use the UNDERLINED, REVERSED, BOLD, or BLINKING phrase, those spaces have the specified character attributes.

75. If you use the PROTECTED phrase on a field that causes the cursor to position past the last column position of the screen, the results are undefined.

76. If you do not use the PROTECTED phrase, an overflow condition is treated according to rules for the Format 1 ACCEPT statement.

**SIZE Phrase (Format 3)**

77. You can use the SIZE phrase only when you also specify the PROTECTED phrase.

78. The SIZE phrase specifies the number of characters in the input field. It allows you to specify fewer or more characters than are specified in the PICTURE clause for `dest-item`.

79. Data input results are undefined if the program generates a value for `prot-size-item` that is zero or negative.

**AUTOTERMINATE Phrase (Format 3)**

80. You can use the AUTOTERMINATE phrase only when you also specify the PROTECTED phrase.

81. The AUTOTERMINATE phrase terminates a protected ACCEPT as soon as the maximum number of characters has been entered. If the maximum number is entered, you should not enter a legal terminator. If you enter the maximum number of characters and a terminator, the terminator is retained in the terminal driver type-
ahead buffer and will terminate the next ACCEPT statement. If you enter fewer than the maximum number of characters, a legal terminator is required to terminate the ACCEPT statement. The maximum number of characters is determined by the same rules that apply to the PROTECTED phrase.

82. If you do not use the AUTOTERMINATE phrase, the ACCEPT statement is terminated when you enter a legal terminator key.

**NO BLANK Phrase (Format 3)**

83. You can use the NO BLANK phrase only when you also specify the PROTECTED phrase.

84. The NO BLANK phrase specifies that the input field is not to be changed until you enter the first keystroke. Once you enter a keystroke, the remainder of the field is filled with spaces or filler (or the default value, if the EDITING and DEFAULT phrases are used) using any character attributes specified on the ACCEPT statement.

**EDITING Phrase (Format 3)**

85. You can use the EDITING phrase only when you also specify the PROTECTED phrase.

86. On OpenVMS systems, the EDITING phrase enables keys to perform field editing functions, described in Table 6.8, “Field Editing Keys for OpenVMS Systems”.

Table 6.8. Field Editing Keys for OpenVMS Systems

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left, Ctrl/D</td>
<td>Move left.</td>
</tr>
<tr>
<td>Right, Ctrl/F</td>
<td>Move right.</td>
</tr>
<tr>
<td>Ctrl/H, F12</td>
<td>Move to beginning of line.</td>
</tr>
<tr>
<td>Ctrl/E</td>
<td>Move to end of line.</td>
</tr>
<tr>
<td>Ctrl/A, F14</td>
<td>Toggle insert and overstrike mode.</td>
</tr>
<tr>
<td>Ctrl/K</td>
<td>Erase to end of line.</td>
</tr>
<tr>
<td>Ctrl/U</td>
<td>Erase to beginning of line.</td>
</tr>
<tr>
<td>Ctrl/M, TAB, CR</td>
<td>Terminate input.</td>
</tr>
<tr>
<td>Ctrl/Z</td>
<td>End-of-file termination.</td>
</tr>
</tbody>
</table>

On Tru64 UNIX systems, the EDITING phrase enables the keys described in Table 6.9, “Field Editing Keys for Tru64 UNIX Systems”.

Table 6.9. Field Editing Keys for Tru64 UNIX Systems

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left, Ctrl/B</td>
<td>Move left</td>
</tr>
<tr>
<td>Right, Ctrl/F</td>
<td>Move right</td>
</tr>
<tr>
<td>Ctrl/H</td>
<td>Erase previous character</td>
</tr>
<tr>
<td>F12, Ctrl/A</td>
<td>Move to beginning of line</td>
</tr>
<tr>
<td>Ctrl/E</td>
<td>Move to end of line</td>
</tr>
<tr>
<td>F14, Ctrl/T</td>
<td>Toggle insert and overstrike mode</td>
</tr>
<tr>
<td>Ctrl/K</td>
<td>Erase to end of line</td>
</tr>
<tr>
<td>Ctrl/U</td>
<td>Erase to beginning of line</td>
</tr>
<tr>
<td>Ctrl/M, TAB, CR</td>
<td>Terminate input.</td>
</tr>
<tr>
<td>Ctrl/D</td>
<td>End of file termination</td>
</tr>
</tbody>
</table>
87. In insert editing, each new character is entered where the cursor indicates and the cursor moves to the right. The rest of the field also moves one character position to the right. A character moved beyond the last character position of the field is lost. The delete key causes the character to the left of the cursor to be deleted. The rest of the field moves one character position to the left, and space is inserted in the last position of the field.

In overstrike editing, each new character replaces the one where the cursor is positioned and the cursor moves to the right. The delete key causes the character to the left of the cursor to be replaced by a space and the cursor moves to the left.

88. When program execution begins, overstrike or insert editing is used, according to your terminal's current setting. The same editing mode is used for all ACCEPT statements with the EDITING phrase until the next use of the switch-mode function.

89. The EDITING phrase affects the syntax and semantics of other phrases. For more information, see the sections on NO BLANK, FILLER, CONVERSION, NO ECHO, DEFAULT, CURRENT VALUE, and CONTROL KEY.

**FILLER Phrase (Format 3)**

90. You can use the FILLER phrase only when you also specify the PROTECTED phrase.

91. The FILLER phrase initializes each character position of the input field with the character specified in `prot-fill-lit`. As you enter characters, the filler characters are replaced by your input. If you strike the delete key after you have entered data, the position made available by the delete operation is refilled with the character specified in `prot-fill-lit`. When you terminate the ACCEPT operation, any remaining filler characters are replaced by space characters.

92. When you use the FILLER phrase with the NO BLANK phrase, the input field is filled with the character specified in `prot-fill-lit`, after you have entered the first character.

93. The FILLER phrase is not allowed to be used with the EDITING phrase. If both are present, the FILLER phrase is ignored.

**NO ECHO Phrase (Format 3)**

94. The NO ECHO phrase suppresses the display of input characters on the screen.

95. When you do not use the NO ECHO phrase, valid input characters are displayed on the screen as they are typed.

96. When the EDITING phrase is used, the field editing functions still take place, but the display field is not modified.

**DEFAULT Phrase (Format 3)**

97. The DEFAULT phrase specifies default input values when no characters are entered from the terminal. Null input is signaled by entering a legal terminator key that is not preceded by data. (See the general rules for the CONTROL KEY phrase.)

98. When the null input condition occurs, `def-src-lit` or the value of `def-src-item` is moved to `dest-item` according to the MOVE rules. When the move occurs, the specified default value is not displayed on the terminal screen.

99. Conversion of the DEFAULT item will occur if CONVERSION is specified.

100. When the EDITING phrase is used, the default value is displayed in the input field. The value can be blank-filled on the right or truncated, depending on the relative lengths of the default value and the input field.

101. When the EDITING phrase is used and a terminator is entered, the contents of the input field are moved to `dest-item` according to the MOVE rules.
CURRENT VALUE Phrase (Format 3)

The CURRENT VALUE phrase can be used only when you specify the DEFAULT phrase.

The CURRENT VALUE phrase specifies that the default input value is the initial value of the ACCEPT destination item.

The value of the ACCEPT destination item is the same as it was before the execution of the ACCEPT statement if all the following conditions exist:

- You specify the CURRENT VALUE phrase.
- The EDITING phrase is not used.
- The default path is taken.

When you use the EDITING phrase, dest-item can be alphabetic, alphanumeric, or non-floating-point numeric. In this case, the input field is always updated to be what is on the screen. It cannot be numeric edited, alphanumeric edited, COMP-1, or COMP-2; if it is, the program will ignore the DEFAULT and CURRENT VALUE phrases.

CONTROL KEY Phrase (Format 3)

When you use the CONTROL KEY phrase in Format 3, the operator must terminate data input with a legal terminator key. Ctrl/Z is a legal terminator if ON EXCEPTION or AT END is specified.

When you do not use the CONTROL KEY phrase in Format 3, the operator can terminate data input only with RETURN or TAB.

When the EDITING phrase is used, the keys which invoke field editing functions do not terminate the ACCEPT statement and are not stored in key-dest-item.

Format 4

When any key other than a valid control key is entered: (1) the contents of key-dest-item do not change, and (2) the terminal bell rings. This occurs until a proper control key is entered.

Format 5

The end-of-file indication is considered a normal, successful termination.

The data in each field is converted and validated as you leave the field. The updated value is displayed in the field if the SECURE clause is not specified.

Conversion occurs in the following instances:

- If the field is not numeric and the JUSTIFIED clause is specified, the data is right-justified.
- If the field is numeric or numeric edited, the data is formatted according to the PICTURE, SIGN, and BLANK WHEN ZERO clauses. This formatting is always successful because only the following characters from the data are formatted:
  - 0–9
  - Period (.), if DECIMAL POINT IS COMMA is not specified
  - Comma (,), if DECIMAL POINT IS COMMA is specified
  - Sign (+, −, DB, db, CR, or cr)
Note that only the first occurrence of the period, comma and sign is accepted; multiple occurrences are ignored. Also, to the left and right of the decimal point excess leftmost digits and excess rightmost digits are truncated, respectively.

Validation occurs when the FULL or REQUIRED clauses are specified.

112. The default value for each field is displayed when the operator enters the field for the first time during an ACCEPT statement. The default value is determined as follows:

- If the USING clause is specified, the default is the current value of the USING item.
- If the TO and FROM clauses are specified, the default is the current value of the FROM item.
- If only the TO clause is specified:
  - The default is ZEROES (or SPACES if BLANK WHEN ZERO is specified) for numeric and numeric edited items.
  - The default is SPACES for alphabetic, alphanumeric, and alphanumeric edited items.

113. If the operator types a terminator key prior to entering data in every field, the default value for each untouched field is moved to the field's destination item.

114. The operator is limited to entering the number of characters specified by the PICTURE clause. If the operator attempts to type beyond the rightmost position of the field: (1) the cursor remains on the position to the right of the rightmost position and (2) the last character of the field is overwritten with the new character.

115. There are special keys that allow the operator to edit data within a field and to move among the fields within a screen. Except where noted otherwise, the operator is allowed to move among the fields in the order in which the fields are defined within the Screen Description Entry.

The keys defined on OpenVMS Alpha and I64 systems are described in Table 6.10, “SCREEN SECTION Keys for OpenVMS Alpha and I64 Systems”.

**Table 6.10. SCREEN SECTION Keys for OpenVMS Alpha and I64 Systems**

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left,</td>
<td>Move left (if not at beginning of field, move left within field; if at beginning of field, move to previous field).</td>
</tr>
<tr>
<td>Ctrl/B,</td>
<td></td>
</tr>
<tr>
<td>Ctrl/D</td>
<td></td>
</tr>
<tr>
<td>Right,</td>
<td>Move right (if not at end of field, move right within field; if at end of field, move to next field).</td>
</tr>
<tr>
<td>Ctrl/F</td>
<td></td>
</tr>
<tr>
<td>Up</td>
<td>Move to the nearest field that is positioned above the current cursor position; this movement ignores the order in which fields are defined within the Screen Description Entry and is based simply on the location of items on the screen.</td>
</tr>
<tr>
<td>Down</td>
<td>Move to the nearest field that is positioned below the current cursor position; this movement ignores the order in which fields are defined within the Screen Description Entry and is based simply on the location of items on the screen.</td>
</tr>
<tr>
<td>Ctrl/P,</td>
<td>Move to previous field.</td>
</tr>
<tr>
<td>Ctrl/L</td>
<td></td>
</tr>
<tr>
<td>TAB,</td>
<td>Move to next field.</td>
</tr>
</tbody>
</table>
### Table 6.11. SCREEN SECTION Keys for Tru64 UNIX Systems

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control/N,</td>
<td>Move to beginning of line (if at beginning of line within a multiple-line</td>
</tr>
<tr>
<td>Control/I</td>
<td>field, move to beginning of previous line).</td>
</tr>
<tr>
<td>Control/H,</td>
<td>Move to end of text (if at end of text within a multiple-line field, move</td>
</tr>
<tr>
<td>Control/W</td>
<td>to end of text on next line).</td>
</tr>
<tr>
<td>Control/E</td>
<td>Toggle insert and overstrike mode (if $ SET NOCONTROL=T).</td>
</tr>
<tr>
<td>Control/A,</td>
<td>Erase to end of line (always performed in insert mode).</td>
</tr>
<tr>
<td>Control/T</td>
<td>Erase to beginning of line (always performed in insert mode).</td>
</tr>
<tr>
<td>Control/K</td>
<td>Terminate input.</td>
</tr>
<tr>
<td>Control/U,</td>
<td>Move left (if not at beginning of field, move left within field; if at</td>
</tr>
<tr>
<td>Control/X</td>
<td>beginning of field, move to previous field).</td>
</tr>
<tr>
<td>Control/P,</td>
<td>Move right (if not at end of field, move right within field; if at end of</td>
</tr>
<tr>
<td>Control/L</td>
<td>field, move to next field).</td>
</tr>
<tr>
<td>Control/B,</td>
<td>Move to the nearest field that is positioned above the current cursor</td>
</tr>
<tr>
<td>Control/D</td>
<td>position; this movement ignores the order in which fields are defined</td>
</tr>
<tr>
<td>Control/F</td>
<td>Move to the nearest field that is positioned below the current cursor</td>
</tr>
<tr>
<td>Ctrl/Up</td>
<td>position; this movement ignores the order in which fields are defined</td>
</tr>
<tr>
<td>Ctrl/Down</td>
<td>Move to previous field.</td>
</tr>
<tr>
<td>Ctrl/TAB,</td>
<td>Move to next field.</td>
</tr>
<tr>
<td>Ctrl/Right</td>
<td>Move to the nearest field that is positioned above the current cursor</td>
</tr>
<tr>
<td>Ctrl/Left</td>
<td>position; this movement ignores the order in which fields are defined</td>
</tr>
</tbody>
</table>

The keys defined on Tru64 UNIX systems are described in Table 6.11, “SCREEN SECTION Keys for Tru64 UNIX Systems “.
The description of insert and overstrike editing for the EDITING Phrase (Format 3) also applies here.

### Formats 6, 7, and 8

When a Format 6 ACCEPT statement is specified, the value of *arg-count* is moved to *dest-item*. This represents the number of arguments on the program run command line (see ARGUMENT-NUMBER in the SPECIAL-NAMES paragraph in Chapter 4, Environment Division).

When the current argument position indicator is zero, it refers to the zeroth command line argument, in other words the command that invoked the COBOL program.

When a Format 7 ACCEPT statement is specified, the value of the command line argument indicated by the current argument position indicator is moved to *dest-item* (see ARGUMENT-VALUE in the SPECIAL-NAMES paragraph in Chapter 4, Environment Division).

The current argument position indicator is determined by the following:

- In the absence of a Format 4 DISPLAY, the initial value of the current argument position indicator is 1.
- The current argument position indicator is incremented by 1 after execution of a Format 7 ACCEPT statement.

When a Format 8 ACCEPT statement is specified, the value of *envlog-value* is moved to *dest-item* (refer to ENVIRONMENT-VALUE and ENVIRONMENT-NAME in the SPECIAL-NAMES paragraph in Chapter 4, Environment Division). This value represents the value of the environment variable or system logical named by the current ENVIRONMENT-NAME item.

Statement 3 is executed if an attempt is made to read beyond the last argument on the command line, or if the argument does not exist.

Statement 4 is executed if the name of the environment variable or logical has not been set by a Format 5 DISPLAY, or if the environment variable or logical does not exist.

Statement 5 is executed if the exception condition does not exist.

### Technical Notes

#### All Formats

1. On OpenVMS systems, if the data transfer is from a terminal, Ctrl/Z is equivalent to an end-of-file indication.
2. On Tru64 UNIX systems, if the data transfer is from a terminal, Ctrl/D is equivalent to an end-of-file indication.

**Format 1**

3. An ACCEPT statement without the FROM phrase takes input from the default input device (the keyboard). To take input from a file on Tru64 UNIX systems, the environment variable COBOL_INPUT can be used to specify a text file containing input data. To take input from a file on OpenVMS systems, the logical COB $INPUT or SYSSINPUT can be used to specify a text file containing input data.

Alternatively, input device redirection ( `<` ) can be used on Tru64 UNIX systems to name an input file.

4. An ACCEPT statement that includes the FROM phrase transfers data from the file-device-name associated with the SPECIAL-NAMES paragraph description of input-source.

5. On OpenVMS systems, the object of a logical name is not necessarily a device. Therefore, no open mode is implied. As a result, input-source can be associated with any device-name in the SPECIAL-NAMES paragraph. For example, input-source can refer to PAPER-TAPE-PUNCH as well as PAPER-TAPE-READER.

6. An end-of-file indication during ACCEPT statement execution with an AT END phrase causes control to transfer to the AT END imperative statement.

7. An end-of-file indication during ACCEPT statement execution without the AT END phrase is an error. The program terminates abnormally.

8. The ACCEPT statement fills dest-item with spaces if the input is an empty record (for example, a carriage return only).

9. On Tru64 UNIX, you can enter a maximum of 256 characters during a Format 1 ACCEPT statement.

**Formats 3 and 4**

- The VSI extensions to the ACCEPT and DISPLAY statements support data input and display only on VT100 and later terminal types, including emulators of these terminal types.

- On OpenVMS Alpha and I64 systems, control sequences from SMGTERMS.TXT are used to accomplish cursor positioning, screen erasure, and video attributes. Refer to the chapter on support for non VSI terminals of the OpenVMS RTL Screen Management (SMG$) Manual if you wish to customize SMGTERMS.TXT.

- You should accept data only from input fields that are within screen boundaries. That is, the terminal operator should see all the characters entered (assuming the NO ECHO, CONVERSION, and PROTECTED phrases are not specified). If you accept data from an input field that positions the cursor outside screen boundaries, the result is not an error condition, but your program might produce unexpected results.

Values for screen boundaries depend on the terminal attributes. Refer to the appropriate terminal user's guide for more information on screen boundaries.

- Line positioning can be a one- or two-step process. The first (or only) step is absolute positioning, which is using the value of line-num or line-id to determine the line position. The second step is relative positioning, which is adding the value of plus-num to line-id to determine the line position. Relative positioning beyond the bottom line of the current screen results in scrolling.

For example, suppose that the screen for which you are programming has a maximum of 24 lines and you need to scroll the screen up one line before accepting data. The following sample statements illustrate how to use relative positioning to accomplish this operation (assume ITEMA has a value of 14, and the current line position is 20):

```
ACCEPT DEST-EXAMPLE FROM LINE NUMBER PLUS 5.
ACCEPT DEST-EXAMPLE FROM LINE NUMBER ITEMA PLUS 11.
```

The following sample statements would produce undefined results because absolute line positioning is beyond the bottom of the screen (assume ITEMB has a value of 25):
ACCEPT DEST-EXAMPLE FROM LINE NUMBER 25.
ACCEPT DEST-EXAMPLE FROM LINE NUMBER ITEMB.
ACCEPT DEST-EXAMPLE FROM LINE NUMBER ITEMB PLUS 0.

The last ACCEPT statement illustrates that use of the PLUS option does not necessarily mean that scrolling will always occur. Absolute line positioning always occurs before the relative positioning specified by the PLUS option. In this case, line-id (ITEMB) has a value of 25. Therefore, the line position is outside the screen boundary before the PLUS option executes, and program results are undefined.

- When you use the CONTROL KEY phrase, key-dest-item stores the terminator key code. The VSI COBOL User Manual contains information on these key code values in its description of programming video forms.

- In Formats 3 and 4, the maximum number of characters in dest-item or key-dest-item is 1024.

- When you use the CONTROL KEY phrase with the PROTECTED WITH AUTOTERMINATE phrase, and the maximum number of characters is entered to terminate the ACCEPT statement, key-dest-item is filled with spaces.

- The ALPHABET clause has no effect on the CONVERSION clause for either ACCEPT or DISPLAY.

- Unexpected behavior can occur when an ACCEPT statement with the EDITING, PROTECTED, and DEFAULT IS CURRENT phrases and without the CONVERSION phrase is executed. The behavior occurs when a numeric data item has a negative scale factor or is signed. To avoid this behavior, it is suggested that the CONVERSION phrase be used in these circumstances.

Additional References

- SPECIAL-NAMES section in Chapter 4, Environment Division
- VSI COBOL User Manual sections on using command-line arguments and environment variables
- Section 6.1.4: Scope of Statements
- DISPLAY
- MOVE statement

Examples

In the following examples, the character s represents a space. The examples assume that the time is just after 2:15 p.m. on October 7, 1992. The Environment and Data Divisions contain the following entries:

SPECIAL-NAMES.
CONSOLE IS IN-DEVICE.
DATA DIVISION.
01 ITEMA PIC X(6).
01 ITEMB PIC 99V99.
01 ITEMC PIC 9(8).
01 ITEMD PIC 9(5).
01 ITEME PIC 9(6).
01 ITEMF PIC 9.
01 ITEMG COMP-1.
01 ITEMH PIC S9(5) COMP.
1. ACCEPT ITEMA.

<table>
<thead>
<tr>
<th>Input</th>
<th>ITEMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPUTER</td>
<td>COMPUT</td>
</tr>
</tbody>
</table>
### Procedure Division

#### Input
<p>|</p>
<table>
<thead>
<tr>
<th>ITEMA</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX</td>
<td>VAXsss</td>
</tr>
<tr>
<td>12.6</td>
<td>12.6ss</td>
</tr>
</tbody>
</table>

2. ACCEPT ITEMB FROM IN-DEVICE.

<table>
<thead>
<tr>
<th>Input</th>
<th>ITEMB</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>1623</td>
<td>1623</td>
<td>16.23</td>
</tr>
<tr>
<td>4</td>
<td>4sss</td>
<td>Invalid data</td>
</tr>
<tr>
<td>60000</td>
<td>6000</td>
<td>60.00</td>
</tr>
<tr>
<td>-1.2</td>
<td>-1.2</td>
<td>Invalid data</td>
</tr>
<tr>
<td>1.23</td>
<td>1.23</td>
<td>Invalid data</td>
</tr>
<tr>
<td>COMPUTER</td>
<td>COMP</td>
<td>Invalid data</td>
</tr>
</tbody>
</table>

3. ACCEPT ITEMB WITH CONVERSION.

<table>
<thead>
<tr>
<th>Input</th>
<th>ITEMB</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>1623</td>
<td>1623</td>
<td>16.23</td>
</tr>
<tr>
<td>4</td>
<td>4sss</td>
<td>04.00</td>
</tr>
<tr>
<td>60000</td>
<td>6000</td>
<td>60.00</td>
</tr>
<tr>
<td>-1.2</td>
<td>-1.2</td>
<td>01.20</td>
</tr>
<tr>
<td>1.23</td>
<td>1.23</td>
<td>01.23</td>
</tr>
<tr>
<td>COMPUTER</td>
<td>COMP</td>
<td>Invalid data</td>
</tr>
</tbody>
</table>

#### STATEMENT

<table>
<thead>
<tr>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT ITEME FROM DATE. ITEME = 921007</td>
</tr>
<tr>
<td>ACCEPT ITEMC FROM TIME. ITEMC = 14150516 (OpenVMS and Tru64 UNIX)</td>
</tr>
<tr>
<td>ACCEPT ITEMD FROM DAY. ITEMD = 92280</td>
</tr>
<tr>
<td>ACCEPT ITEMF FROM DAY-OF-WEEK. ITEMF = 3</td>
</tr>
<tr>
<td>ACCEPT ITEMA FROM TIME. ITEMA = 141505</td>
</tr>
<tr>
<td>ACCEPT ITEME FROM TIME. ITEME = 150516</td>
</tr>
<tr>
<td>ACCEPT ITEMD FROM DAY-OF-WEEK. ITEMD = 00003</td>
</tr>
</tbody>
</table>

4. ACCEPT ITEMH WITH CONVERSION.

<table>
<thead>
<tr>
<th>Input</th>
<th>Result Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>.123E-02</td>
<td>0.00123</td>
</tr>
<tr>
<td>-12.3E+02</td>
<td>-1230</td>
</tr>
<tr>
<td>1004E-07</td>
<td>1.004000E-04</td>
</tr>
</tbody>
</table>

4. ACCEPT ITEMH WITH CONVERSION.

<table>
<thead>
<tr>
<th>Input</th>
<th>Result Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>-44</td>
<td>-44</td>
</tr>
</tbody>
</table>

Additional examples containing VSI extensions to the ACCEPT statement (Formats 3, 4, and 5) are described in the VSI COBOL User Manual. Refer to the description of programming video forms.
Also, examples containing extensions to the ACCEPT statement (Formats 6, 7 and 8) that access command line arguments are described in the VSI COBOL User Manual.

6.9.

ADD

ADD

Function

The ADD statement adds two or more numeric operands and stores the sum in one or more receiving fields.

[num]

is a numeric literal or the identifier of an elementary numeric item.

[result]

is the identifier of an elementary numeric item. However, in Format 2, result can be an elementary numeric edited item. It is the resultant identifier.

[stmt]

is an imperative statement executed when a size error condition has occurred.

[stmt2]

is an imperative statement executed when no on size error condition has occurred.

[group-1]

is the identifier of numeric group item.

[group-2]

is the identifier of numeric group item.

Syntax Rule

CORR is an abbreviation for CORRESPONDING.

General Rules

1. In Format 1, the values of the operands before the word TO are added together. This total is then added to each occurrence of result.

2. In Format 2, the values of the operands before the word GIVING are added. The sum is then stored in each result.

3. In Format 3, data items in group-1 are added to and stored in the corresponding data items in group-2.

Additional References

• Section 6.1.4: Scope of Statements

• Section 6.6.1: Arithmetic Operations

• Section 6.6.2: Multiple Receiving Fields in Arithmetic Statements

• Section 6.6.3: ROUNDED Phrase
• Section 6.6.4: ON SIZE ERROR Phrase
• Section 6.6.5: CORRESPONDING Phrase
• Section 6.6.7: Overlapping Operands and Incompatible Data

Examples

Each of the examples assumes the following data descriptions and initial values:

INITIAL VALUES

```
03 ITEM A PIC 99 VALUE 85.                85
03 ITEMB PIC 99 VALUE 2.                 2
03 ITEM C VALUE "123".
  05 ITEMD OCCURS 3 TIMES              1 2 3
    PIC 9.
```

1. TO phrase: RESULTS

```
ADD 2 ITEMB TO ITEMA.                           ITEMA = 89
```

2. SIZE ERROR clause:

```
ADD 38 TO ITEMA ITEMB                           ITEMA = 85
ITEMB = 40
ON SIZE ERROR
  MOVE 0 TO ITEMB.                            ITEMB = 0
```

(When the SIZE ERROR condition occurs, the value of the affected resultant identifier does not change. The SIZE ERROR condition occurs on ITEMA but not on ITEMB.)

3. NOT ON SIZE ERROR clause:

```
ADD 14 TO ITEMA                                 ITEMA = 99
ON SIZE ERROR
  MOVE 0 TO ITEMB.                           ITEMB = 0
  NOT ON SIZE ERROR
    MOVE 1 TO ITEMB.                         ITEMB = 1
```

(If the SIZE ERROR condition had occurred, the value of ITEMA would have been 85 and ITEMB would have been 0.)

4. Multiple receiving fields:

```
ADD 1 TO ITEMB ITEMD (ITEMB).                   ITEMB = 3
ITEMD (3) = 4
```

(The operations proceed from left to right. Therefore, the subscript for ITEMD is evaluated after the addition changes its value.)

5. GIVING phrase:

```
ADD ITEMB ITEMD (ITEMB) GIVING ITEMA.           ITEMA = 4
```

6. END-ADD:

```
IF ITEMB < 10
  ADD 7 ITEMB TO ITEMD (ITEMB)                 ITEMD (3) = 4
  ON SIZE ERROR
    MOVE 0 TO ITEMB                            ITEMB = 0
END-ADD
```
ADD 1 TO ITEMB.

(ITEMB = 1)

(The first ADD terminates with END-ADD. If the SIZE ERROR condition had not occurred, the second ADD statement would have executed anyway; the value of ITEMB would have been 3.)

**ALTER**

**ALTER**

**Function**

The ALTER statement changes the destination of a GO TO statement.

[proc]

is the name of a paragraph that contains one sentence: a GO TO statement without the DEPENDING phrase.

[new-proc]

is a procedure-name.

**General Rules**

1. The ALTER statement changes the destination of the GO TO statement in *proc*.

2. When the changed GO TO executes, it transfers control to *new-proc* instead of the procedure it previously referred to.

   However, when the GO TO statement is in an independent segment (segment-number 50 to 99), the GO TO statement could return to its initial state under some circumstances.

3. A GO TO statement in a section with a segment-number greater than 49 cannot be changed by an ALTER statement in a section with a different segment-number.

**Additional References**

- Section 6.7: Segmentation
- GO TO statement

**Examples**

The examples assume the following Procedure Division code:

PROC-AA.
    DISPLAY "PROC-A".
PROC-A.
    GO TO PROC-BB.
PROC-BB.
    DISPLAY "PROC-B".
PROC-B.
    GO TO PROC-DD.
PROC-CC.
    DISPLAY "PROC-C".
PROC-C.
    GO TO PROC-FF.
PROC-DD.
DISPLAY "PROC-D".
PROC-D.
   GO TO PROC-CC.
PROC-EE.
   DISPLAY "PROC-E".
PROC-E.
   GO TO.
PROC-FF.
   DISPLAY "PROC-F".
PROC-F.
   EXIT.

1. As written.

   Output
   PROC-A
   PROC-B
   PROC-D
   PROC-C
   PROC-F

2. ALTER PROC-A TO PROC-EE PROC-E TO PROC-CC.

   Output
   PROC-A
   PROC-E
   PROC-C
   PROC-F

3. ALTER PROC-D TO PROC-EE PROC-C TO PROC-AA.

   Output
   PROC-A
   PROC-B
   PROC-D
   PROC-E
   error at PROC-E

CALL

Function

The CALL statement transfers control to another program in the executable image.

[ prog-name ]

is a nonnumeric literal or the identifier of an alphanumeric data item. It is the name of the program to which
control transfers.

[ arg ]

is the argument. It identifies the data that is available to both the calling and called programs. It is any data item
described in the File Section, Working-Storage Section, or Linkage Section, or it is a nonnumeric literal. It must
not be a function-identifier.

[ function-res ]
is the identifier of an elementary integer numeric data item with COMP, COMP-1, or COMP-2 usage and no scaling positions. function-res can be subscripted, and it can be qualified. When control returns to the calling program, function-res can contain a function result.

[stmt]

is an imperative statement executed for an on exception or an overflow condition.

[stmt2]

is an imperative statement executed for a not on exception or a not on overflow condition.

**Syntax Rules**

1. *prog-name* must be from 1 to 31 characters long. It can contain the characters "A" to "Z", "a" to "z", "0" to "9", and hyphen (-), dollar sign ($), and underline (_).

2. *prog-name* is the entry-point in the called program. For COBOL programs, *prog-name* is the program-name specified in the PROGRAM-ID paragraph.

3. The same *arg* can appear more than once in the USING phrase.

4. The maximum number of arguments is 255.

5. If there is no initial argument-passing mechanism (REFERENCE, VALUE, CONTENT, or, for DESCRIPTOR), BY REFERENCE is the default.

6. An argument-passing mechanism applies to every *arg* following it until a new mechanism (if any) appears.

7. The CALL statement has a USING phrase only if a USING phrase is in the Procedure Division header of the called program. Both USING phrases must have the same number of arguments.

8. If *arg* is a nonnumeric literal, only BY REFERENCE, BY CONTENT, or for OpenVMS systems, BY DESCRIPTOR can be used.

9. OMITTED, a reserved word, indicates the absence of a specific argument. OMITTED does not change the default argument-passing mechanism; it generates BY VALUE 0 for the omitted argument.

10. If the argument-passing mechanism is BY VALUE, *arg* must be either: (a) an integer numeric literal in the range -2**31 to +2**31-1, (b) a COMP-1 data item, or (c) a word or longword integer COMP data item.

**General Rules**

1. The program whose name is specified by *prog-name* is the called program. The program containing the CALL statement is the calling program.

2. When the CALL statement executes, the contents of *prog-name* are interpreted as follows:
   - Hyphens are treated as underline characters.
   - Lowercase letters are treated as uppercase (See the Technical Notes relating to case sensitivity later in this section).
   - Leading and trailing spaces and tab characters are ignored.

3. The CALL statement transfers control to the called program.

4. Two or more programs in the run unit can have the same *prog-name*. The scope of names conventions for program-names resolve the CALL statement references to duplicate *prog-names*. (See the section on Conventions for Resolving Program-Name References.)
5. If `prog-name` is an identifier, the CALL statement can transfer control only to VSI COBOL programs.

6. The ON EXCEPTION phrase is interchangeable with the ON OVERFLOW phrase.

7. If `prog-name` is not in the executable image and there is an ON EXCEPTION phrase, any NOT ON EXCEPTION phrase is ignored, `stmt` executes, and control is transferred to the end of the CALL statement.

8. If `prog-name` is in the executable image, and there is an ON OVERFLOW phrase or ON EXCEPTION phrase, both phrases are ignored. Control is transferred either to the end of the CALL statement or, if NOT ON EXCEPTION is specified, to `stmt2`. After `stmt2` executes, control is transferred to the end of the CALL statement.

9. If `prog-name` is not in the executable image and there is no ON EXCEPTION phrase, an error condition exists; the program terminates abnormally.

10. If the called program does not have the initial attribute, it, and each program directly or indirectly contained in it, is in its initial state: (a) the first time it is called in an image, and (b) the first time it is called after a CANCEL to the called program.

    On all other entries, the state of the called program is the same as when it was last exited. The program state includes internal data.

11. If the called program has the initial attribute, it, and each program directly or indirectly contained in it, is in its initial state every time it is called.

12. Files associated with a called program's internal file connectors are not in the open mode:
    - The first time the program is called
    - The first time the program is called after execution of a CANCEL statement referring to the program
    - Every time the program is called, if it has the initial attribute

    On all other entries, the status and positioning of such files in a called program are the same as when the program was last exited.

13. The process of calling a program or exiting from a called program does not alter the status or positioning of a file associated with any external file connector.

14. The arguments' order of appearance in the USING phrases of the CALL statement and the called program's Procedure Division header determine correspondence between the data-names used by the calling and called programs. Data-names correspond by position in the USING phrase, not by name.

    No correspondence exists for index-names. If a table is passed as an argument, the index associated with that table in the called program will be the one specified in the INDEXED BY phrase in the called program, not the index specified in the calling program.

15. The arguments in the CALL statement USING phrase are made available to the called program when the CALL executes.

16. Called programs can contain CALL statements. However, a called program must not execute a CALL statement that directly or indirectly calls the calling program.

17. The CALL statement can make data available to the called program by four argument-passing mechanisms:
    - **REFERENCE**—The address of (pointer to) `arg` is passed to the called program. This is the default mechanism: arguments are passed BY REFERENCE if there is no explicit mechanism in the CALL statement.
    - **CONTENT**—The address of a temporary data item that contains the contents of `arg` is passed to the called program.
• VALUE—The value of arg is passed to the called program. If arg is a data-name, its description in the Data Division can be either:
  • COMP usage with no scaling positions; the picture can specify no more than nine digits
  • COMP-1 usage

Note that OMITTED, an VSI COBOL reserved word, is equivalent to BY VALUE 0 and can be used in place of that BY VALUE argument-passing mechanism.

18.

19. If the BY REFERENCE phrase is either specified or implied for a parameter, the called program references the same storage area for the data item as the calling program. This mechanism ensures that the contents of the parameter in the calling program are always identical with the contents of the parameter in the called program.

20. If the BY CONTENT phrase is either specified or implied for a parameter, a copy of arg is moved to a temporary memory location, and the address of the temporary memory location is passed to the called program. This mechanism ensures that the called program cannot change the original contents of arg. However, the called program can change the value of the temporary memory location.

21.

The data description of each arg in the calling program must be identical to each arg in the called program. The compiler does not convert, extend, or truncate any arg passed to a called program.

22. On Alpha and I64 systems, if the GIVING phrase of the CALL statement is not specified, the function result is made available in the RETURN-CODE special register when control returns to the calling program.

23. If the GIVING phrase is specified, the function result is made available in function-res when control returns to the calling program.

Technical Notes

• On Alpha and I64 systems, because the reserved word RETURN-CODE is one of the X/Open reserved words, you cannot use the reserved words compiler option with the noxopen setting if you want to use the RETURN-CODE special register.

For more information on the relationship between the GIVING phrase and the RETURN-CODE special register, see Table 6.7, “Relation of GIVING Phrase to RETURN-CODE Special Register (Alpha, I64)” in this chapter.

• On Tru64 UNIX systems, the linker is case sensitive, whereas the COBOL language is case insensitive. When prog-name in a CALL statement is a literal, and you are calling a program in a case-sensitive language (such as C), you might need to use a form of the names option when you compile.

If you do not specify the names option on the command line, the default setting is lowercase, which causes the VSI COBOL compiler to force all external names to be lowercase. Hence, there is no problem when you call a C program whose name contains no uppercase letters. If the name consists of all uppercase letters, use the uppercase setting of the names option. If it is mixed-case (for example, "Cprog") use the as_is setting. When you use names as_is, only literals in the CALL program name are affected.

• CALL identifier (CALL data name) requires that all modules be specified to link the run unit. Since there are no link-time references to routines to be called with CALL identifier, the linkers on OpenVMS and Tru64 UNIX do not resolve these references at link time. Instead, the references are dynamically resolved at run-time using modules which have been explicitly linked into the run unit.

Additional References

• Section 6.8: General Formats and Rules for Statements
• PROGRAM-ID paragraph in Chapter 3, Identification Division
Examples

1. Passing arguments by reference:

   CALL "DATERTN" USING ITEMA ITEMB ITEMC.

2. On OpenVMS, mixing argument-passing mechanisms: Reference arguments are ITEMA, ITEMD, and "PAYROLL". Descriptor arguments are ITEMB, ITEMC, ITEMD, "TOTALS", and ITEMF. The value arguments are ITEME and "995.99". ITEMD is passed twice—by reference and by descriptor. The content arguments are ITEMG and "SUMMARY FLAG".

   CALL "NEWPROG" USING ITEMA
   BY DESCRIPTOR ITEMB ITEMC "TOTALS"
   BY REFERENCE ITEMD "PAYROLL"
   BY VALUE ITEME 995.99
   BY DESCRIPTOR ITEMD ITEMF
   BY CONTENT ITEMG "SUMMARY FLAG".

3. Mixing argument-passing mechanisms: Reference arguments are ITEMA, ITEMD, and "PAYROLL". The value arguments are ITEME and "995.99". The content arguments are ITEMG and "SUMMARY FLAG".

   CALL "NEWPROG" USING ITEMA
   BY REFERENCE ITEMD "PAYROLL"
   BY VALUE ITEME 995.99
   BY CONTENT ITEMG "SUMMARY FLAG".

4. Calling a program whose name is selected at run time:

   MOVE "PROG009" TO PROG-TO-CALL.
   .
   .
   CALL PROG-TO-CALL USING ITEMA.

5. Receiving a function result:

   CALL "PROG010" USING ITEMA ITEMB "XYZ" GIVING ITEMC.

CANCEL

Function

The CANCEL statement returns the named program to its initial state.
is a nonnumeric literal or the identifier of an alphanumeric data item. It contains the program-name of the program to be canceled.

**Syntax Rules**

1. `prog-name` must be from 1 to 31 characters long. It can contain the characters A to Z, a to z, 0 to 9, dollar sign ($), hyphen (-), and underline (_).

2. `prog-name` must be the name of an VSI COBOL program.

**General Rules**

1. Two or more programs in the run unit can have the same `prog-name`. The scope of names conventions for `prog-names` resolve the CANCEL statement references to duplicate `prog-names`. (See the section on Conventions for Resolving Program-Name References.)

2. Using the scope of names conventions (See Section 6.2.6, “Scope of Names”), if `prog-name` is called again after the CANCEL statement successfully executes, `prog-name`, and all programs contained within it, are in their initial state.

3. `prog-name` must be callable by the program that contains the CANCEL statement.

4. The program named by `prog-name` must not refer directly or indirectly to any program that: (a) has been called, and (b) has not yet executed an EXIT PROGRAM statement.

5. When the CANCEL statement executes, the contents of `prog-name` are interpreted as follows:
   - Hyphens are treated as underline characters.
   - Lowercase letters are treated as uppercase.
   - Leading and trailing spaces and tab characters are ignored.

6. A called program can be canceled in three ways:
   - By being named in a CANCEL statement
   - When the executable image ends
   - When an EXIT PROGRAM statement executes if the program has the initial attribute

7. When canceling a program these items do not change: (a) the contents of its data items in external data records, and (b) the status and positioning of a file associated with any external file connector.

8. During the execution of a CANCEL statement, an implicit CLOSE statement without any optional phrases executes for each file in the open mode that is associated with an internal file connector in `prog-name`.

**Additional References**

- PROGRAM-ID paragraph in Chapter 3, Identification Division
- Section 6.1.4: Scope of Statements
- Section 6.2.6: Scope of Names
- CALL statement
- VSI COBOL User Manual cobcall, cobcancel, cobfunc references

**Examples**

1. CANCEL “PROG10”.

277
CLOSE

Function

The CLOSE statement ends processing of reels (or units) and files. It can also perform rewind, lock, and removal operations.

[file-name]

is the name of a file described in the Data Division. It cannot be a sort or merge file.

Syntax Rules

1. The REEL or UNIT phrase can be used only for sequential and line sequential files.
2. The words REEL and UNIT are equivalent.

General Rules

1. A CLOSE statement can execute only for an open file.
2. Executing a CLOSE statement updates the value of the FILE STATUS data item associated with the file.
3. The TERMINATE statement must be executed before a CLOSE statement can reference a report file.
4. If an optional file is not present, standard end-of-file processing does not occur.
5. The WITH NO REWIND and FOR REMOVAL phrases have no effect at execution time if they do not apply to the file's storage medium, except as specified in General Rule 2.
6. When the CLOSE statement applies to an output or extend file described with the LINAGE clause, end-of-page processing occurs before the file is closed.
7. After successful CLOSE statement execution (without the REEL or UNIT phrase), the file's record area is no longer available. After unsuccessful execution, record area availability is undefined.
8. After successful CLOSE statement execution (without the REEL or UNIT phrase), the file is no longer: (a) in the open mode or (b) associated with the file connector.
9. If the CLOSE statement has more than one file-name, the statement executes as if there were a separate CLOSE statement for each file-name.
10. In the file-sharing environment, CLOSE statement execution unlocks all locks for file-name.
11. If both the REEL/UNIT and WITH NO REWIND phrases are specified in the same CLOSE statement, the WITH NO REWIND phrase is ignored.
12. To show the effects of CLOSE statements, all files are categorized as follows:
   - Nonreel: a file for which the concepts of rewind and reel have no meaning because of its input or output medium (for example, a terminal device)
   - Sequential single-reel: a sequential file contained entirely on one reel
   - Sequential multi-reel: a sequential file contained on more than one reel
• **Nonsequential**: a file with other than sequential organization, whose medium is on a mass storage device

13. For files specified with a MULTIPLE FILE TAPE clause the NO REWIND phrase, if any, is ignored.

14. Table 6.12, “Effects of CLOSE Statement Formats on Files by Category” summarizes CLOSE statement results. Symbol definitions follow the table.

Where definitions differ for input, output, and input-output files, separate definitions appear. Otherwise, a definition applies to files in all open modes.

**Table 6.12. Effects of CLOSE Statement Formats on Files by Category**

<table>
<thead>
<tr>
<th>CLOSE Statement Format</th>
<th>File Category</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nonreel</td>
</tr>
<tr>
<td>CLOSE</td>
<td>C</td>
</tr>
<tr>
<td>CLOSE WITH LOCK</td>
<td>C,E</td>
</tr>
<tr>
<td>CLOSE WITH NO REWIND</td>
<td>C,H</td>
</tr>
<tr>
<td>CLOSE REEL</td>
<td>F</td>
</tr>
<tr>
<td>CLOSE REEL FOR REMOVAL</td>
<td>F,D,G</td>
</tr>
</tbody>
</table>

• Previous reels unaffected

For input and input-output files: All reels in the file before the current reel are processed according to the standard reel swap procedure. However, reels controlled by an earlier CLOSE REEL/UNIT statement are not affected. If other reels in the file follow the current reel, they are not processed.

For output files: All reels in the file before the current reel are processed according to the standard reel swap procedure. However, reels controlled by an earlier CLOSE REEL/UNIT statement are not affected.

• No rewind of current reel

The position of the current reel remains the same.

• Close file

The file is closed.

• Reel/unit removal

The current reel rewinds and is logically removed from the run unit. However, the run unit can access the reel again in its proper order of reels in the file. To do this, the executable image must subsequently execute the following:

• A CLOSE statement without the REEL/UNIT phrase for the file

• An OPEN statement for the file

• File lock

The executable image cannot open the file again in its current execution.

• Close reel/unit

For input and input-output files, if the current reel is the last or only reel for the file:
• A reel swap does not occur.
• The Current Volume Pointer remains the same.
• The File Position Indicator denotes that there is no next logical record.

If another reel follows the current reel for the file:
• A reel swap occurs.
• The Current Volume Pointer points to the next reel for the file.
• The File Position Indicator points to the next record in the file. If there are no records for the current volume, another reel swap occurs.

For output files (reel/unit media), a reel swap occurs. The Current Volume Pointer points to the new reel.
Executing the next WRITE statement for the file transfers a logical record to the new reel of the file.

For output files (nonreel/unit media), execution of this statement is considered successful. The file remains in the open mode and no action takes place, except as specified in General Rule 2.

• Rewind
  The current reel (or device) is positioned to its physical beginning.

• Optional phrases ignored
  The CLOSE statement is executed as if none of the optional phrases are present.

• Invalid
  This is an invalid combination of CLOSE option and file category. It results in FILE STATUS data item value 30.

**Technical Note**
CLOSE statement execution can result in these FILE STATUS data item values:

<table>
<thead>
<tr>
<th>File Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Successful</td>
</tr>
<tr>
<td>07</td>
<td>CLOSE statement with NO REWIND, REEL/UNIT, or FOR REMOVAL phrase referenced a file on a nonreel/unit medium</td>
</tr>
<tr>
<td>30</td>
<td>Any other CLOSE error</td>
</tr>
<tr>
<td>42</td>
<td>File never opened, already closed, or not currently open</td>
</tr>
</tbody>
</table>

**Additional Reference**
See Section 6.6.8: I-O Status for more information.

**COMPUTE**

**Function**
The COMPUTE statement evaluates an arithmetic expression and stores the result in one or more data items.
[result]
is the identifier of an elementary numeric item or elementary numeric edited item. It is the resultant identifier.

[arithmetic-expression]
is an expression as described in Section 6.4: Arithmetic Expressions.

[statement]
is an imperative statement executed when a size error condition has occurred.

[statement2]
is an imperative statement executed when no size error condition has occurred.

**General Rules**

1. The arithmetic expression is evaluated. Its value then replaces the current value of each occurrence of *result*, from left to right.

2. If the *arithmetic-expression* consists of a single identifier or literal, the COMPUTE statement behaves like a MOVE statement with the single identifier or literal acting as the source operand and each result operand acting as a destination operand.

3. For any *result* specification that includes the word rounded, the value of the expression is rounded before being moved to *result*.

**Additional References**

- Section 6.1.4: Scope of Statements
- Section 6.6.1: Arithmetic Operations
- Section 6.6.2: Multiple Receiving Fields in Arithmetic Statements
- Section 6.6.3: ROUNDED Phrase
- Section 6.6.4: ON SIZE ERROR Phrase
- Section 6.6.7: Overlapping Operands and Incompatible Data

**Examples**

Each of the examples assume these data descriptions and initial values:

**INITIAL VALUES**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>ITEM A</td>
<td>PIC 999V99 VALUE 2.</td>
<td>2.00</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>ITEM B</td>
<td>PIC 999V99 VALUE 3.</td>
<td>3.00</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>ITEM C</td>
<td>PIC 999V99 VALUE 4.</td>
<td>4.00</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>ITEM D</td>
<td>PIC 999V99 VALUE 5.</td>
<td>5.00</td>
<td></td>
</tr>
</tbody>
</table>

**RESULTS**

1. No rounding:

   \[
   \text{COMPUTE ITEM C} = (\text{ITEM A} + 6) ** (0.1 * \text{ITEM D}).
   \]

   \[
   \text{ITEM C} = 2.82
   \]

2. With rounding:

   \[
   \text{COMPUTE ITEM C ROUNDED} = (\text{ITEM A} + 6) ** (0.1 * \text{ITEM D}).
   \]

   \[
   \text{ITEM C} = 2.83
   \]
3. The ON SIZE ERROR phrase:

\[
\begin{align*}
\text{COMPUTE ITEMB} &= (\text{ITEMA} \times \text{ITEMD})^2 \times 3 \\
&= \text{ITEMB} = 3.00 \\
\text{ON SIZE ERROR} \\
\text{MOVE 100 TO ITEMC.} &\quad \text{ITEMC} = 100.00
\end{align*}
\]

4. The NOT ON SIZE ERROR phrase:

\[
\begin{align*}
\text{COMPUTE ITEMB} &= (\text{ITEMA} \times \text{ITEMD})^2 \\
&= \text{ITEMB} = 100.00 \\
\text{ON SIZE ERROR} \\
\text{MOVE 100 TO ITEMC} \\
\text{NOT ON SIZE ERROR} \\
\text{MOVE 200 TO ITEMC.} &\quad \text{ITEMC} = 200.00
\end{align*}
\]

CONTINUE

Function

The CONTINUE statement indicates that no executable statement is present. It causes an implicit control transfer to the next executable statement.

Syntax Rule

The CONTINUE statement can be used wherever a conditional or imperative statement can be used.

General Rule

The CONTINUE statement causes an implicit control transfer to the next executable statement.

Example

READ FILE-A
  INVALID KEY
  CONTINUE.
  MOVE ...

This example shows how CONTINUE can replace an INVALID KEY imperative statement. Control passes to the MOVE statement whether or not the INVALID KEY condition occurs.

DELETE

Function

The DELETE statement logically removes a record from a mass storage file.

[file-name]

is the name of a relative or indexed file described in the Data Division. It cannot be the name of a sequential or line sequential file or a sort or merge file.

[stmt]
is one or more imperative statements executed for an invalid key condition.

Syntax Rules

1. There cannot be an INVALID KEY phrase or a NOT INVALID KEY phrase for a DELETE statement that references a file in sequential access mode.

2. There must be an INVALID KEY phrase if: (a) the file is not in sequential access mode and (b) there is no applicable USE AFTER EXCEPTION procedure.

General Rules

1. The file must be open in I-O mode when the DELETE statement executes.

2. For a file in sequential access mode, a successfully executed READ statement must be the last input-output statement executed for the file before the DELETE statement. The I/O system logically removes the record that the READ statement accessed.

3. For a relative file in random or dynamic access mode, the I/O system logically removes the record identified by the file's RELATIVE KEY data item. If the file does not contain that record, an invalid key condition exists.

4. For an indexed file in random access mode, the I/O system (logically removes the record identified by the file's primary record key data item. If the file does not contain that record, an invalid key condition exists.

5. For an indexed file in dynamic access mode, the behavior depends on the DUPLICATES phrase of the RECORD KEY clause of the SELECT statement. If the primary key allows duplicates, Rule 2 applies. If the primary key does not allow duplicates, Rule 4 applies.

6. After successful DELETE statement execution, the identified record has been logically removed from the file. It is no longer accessible.

7. DELETE statement execution does not affect the contents of the record area. It also does not affect the contents of the data item referred to in the DEPENDING ON phrase of the file's RECORD clause.

8. For sequential access files, DELETE statement execution does not affect the File Position Indicator.

9. For dynamic access files, the File Position Indicator can point to the record to be deleted before the DELETE statement executes. In this case, once the DELETE statement executes, the File Position Indicator:
   • Points to a relative file's next existing record
   • Points to an indexed file's next existing record, as established by the Key of Reference
   • Indicates the at end condition if the file has no next record

   In all other cases, the File Position Indicator is not affected by the execution of a DELETE statement.

10. DELETE statement execution updates the value of the FILE STATUS data item for the file.

11. If there is an applicable USE AFTER EXCEPTION procedure, it executes whenever an input or output condition occurs that would result in a nonzero value in the first character of a FILE STATUS data item. However, it does not execute if the condition is invalid key, and there is an INVALID KEY phrase. If the condition is not invalid key and no applicable USE AFTER EXCEPTION Declarative procedure exists, the run unit terminates abnormally.

See the rules for the INVALID KEY phrase, Section 6.6.10, “INVALID KEY Phrase”.

Technical Notes
• In a VSI standard file-sharing environment for relative files, records that are manually locked and then deleted retain a lock on the deleted record. This affects a subsequent WRITE from a different access stream (either from a different file connector in the same run-unit or from a file connector in a different run-unit). A WRITE statement under such conditions gets a file status 92 (record locked by another program). All other statements will treat a record that was locked and subsequently deleted in the same manner as a record that was not locked and subsequently deleted.

• DELETE statement execution can result in the following FILE STATUS data item values:

<table>
<thead>
<tr>
<th>File Status</th>
<th>Access Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>All</td>
<td>Successful</td>
</tr>
<tr>
<td>23</td>
<td>Rand, Dyn</td>
<td>Record not in file (invalid key)</td>
</tr>
<tr>
<td>43</td>
<td>Seq</td>
<td>No previous READ or record not locked by prior READ or START</td>
</tr>
<tr>
<td>49</td>
<td>All</td>
<td>File not open, or incompatible open mode</td>
</tr>
<tr>
<td>92</td>
<td>All</td>
<td>Record locked by another program</td>
</tr>
<tr>
<td>30</td>
<td>All</td>
<td>All other permanent errors</td>
</tr>
</tbody>
</table>

• If the current record to be deleted is not locked by the current stream, the delete results in a permanent error. On OpenVMS systems, RMS-STS indicates the record is not locked.

Addional References

• Section 6.1.4: Scope of Statements
• Section 6.6.8: I-O Status
• Section 6.6.10: INVALID KEY Phrase
• OPEN statement
• USE statement

DISPLAY

DISPLAY

Function

The DISPLAY statement transfers low-volume data from the program to the default system output device or to the object of a mnemonic-name. The WITH CONVERSION phrase in Format 1 contains a VSI extension to the DISPLAY statement. The VSI extensions to Formats 2 and 3 are COBOL language additions that facilitate video forms design and data handling.

Format 4 sets a program variable to the current command line argument number (to read with a Format 7 ACCEPT), Format 5 sets the name of an environment variable or system logical, and Format 6 sets the value of an environment variable or system logical.

[src-item]

is a literal or the identifier of a data item. The literal can be any figurative constant.

[arg-position]

is a literal or identifier that specifies the desired argument position (on the run command line). It must be an unsigned integer.
[argument-number]

is a mnemonic name associated with ARGUMENT-NUMBER in the SPECIAL-NAMES paragraph in the Environment Division, representing the current argument position.

[environment-name]

is a mnemonic name associated with ENVIRONMENT-NAME in the SPECIAL-NAMES paragraph in the Environment Division, representing the name of an environment variable or system logical.

[environment-value]

is a mnemonic name associated with ENVIRONMENT-VALUE in the SPECIAL-NAMES paragraph in the Environment Division, representing the contents of the variable associated with the ENVIRONMENT-NAME.

[envlog-name]

references an alphanumeric data item, or is a nonnumeric literal.

[envlog-value]

references an alphanumeric data item, or is a nonnumeric literal.

[output-dest]

is a mnemonic-name defined in the SPECIAL-NAMES paragraph of the Environment Division.

[line-num]

is a numeric literal that specifies a line position on the terminal screen. line-num must be a positive integer. It cannot be zero.

[line-id]

is the identifier of a data item that provides a line position on the terminal screen. It must be a positive integer; it cannot be zero.

[plus-num]

is a numeric literal that increments the current value for line or column position, or that increments the value of line-id or column-id. plus-num can be zero or a positive integer.

[column-num]

is a numeric literal that specifies a column position on the terminal screen. column-num must be a positive integer. It cannot be zero.

[column-id]

is the identifier of a data item that provides a column position on the terminal screen. It must be a positive integer; it cannot be zero.

[screen-name]

is the name of a screen item defined in the SCREEN SECTION of the program.

[stmt]

is an imperative statement executed if an exception condition exists; for Format 6, this means the name of the environment variable or logical has not been set by DISPLAY, or not enough space can be allocated to store the environment variable or logical.

[stmt2]

is an imperative statement executed if the exception condition does not exist.
Syntax Rules

All Formats

1. In a DISPLAY statement, the number of src-item entries cannot exceed 254.

2. Each DISPLAY phrase can be specified only once for any src-item.

Formats 1 and 2

1. The WITH NO ADVANCING phrase can be specified only once per DISPLAY statement. It must be specified last (or just preceding END-DISPLAY, if used) if multiple src-item entries or options are specified in the statement.

Format 1

1. The UPON phrase can be specified only once per DISPLAY statement.

General Rules

Formats 1 and 2

1. The UPON and WITH NO ADVANCING phrases apply to all instances of src-item.

2. All phrases other than UPON and WITH NO ADVANCING apply to the immediately preceding src-item only.

3. If there is a WITH NO ADVANCING phrase, the DISPLAY statement does not transfer any device positioning information after the last src-item value.

CONVERSION Phrase (Formats 1 and 2)

1. The CONVERSION phrase allows you to display data in a field and achieve data conversion, sign, and decimal point placement. How the CONVERSION phrase affects data handling depends on the category of src-item. (Numeric data can be described by any USAGE clause.)

2. Numeric items do not require the CONVERSION phrase to be displayed correctly with conversion if you specify /DISPLAY_FORMATTED (on OpenVMS Alpha and I64) or -display_formatted (on Tru64 UNIX) when you compile. (For more information on the qualifier, refer to the VSI COBOL User Manual.)

3. The CONVERSION phrase or the /DISPLAY_FORMATTED (OpenVMS Alpha and I64) qualifier displays nonnumeric items and numeric edited items without change.

4. The CONVERSION phrase or the /DISPLAY_FORMATTED (OpenVMS Alpha and I64) qualifier displays non-floating-point numeric items according to the following rules:
   - The size of the displayed field is determined from the PICTURE character string.
   - Leading zeroes are displayed only when they immediately precede a decimal point.
   - If the sign is negative, a minus sign (-) is displayed. If the sign is positive, a space character is displayed. If the item is unsigned, no sign position is displayed.
   - Items with DISPLAY usage (for example, PIC 99, or PIC S99V99) are displayed after including a space, when needed, for a decimal point or sign or both.

   If you specify the SIGN TRAILING SEPARATE clause for the data item, the sign is displayed as a trailing sign. Otherwise, the sign is displayed as a leading sign.
• Items other than DISPLAY (for example, PIC 99 COMP, or PIC S9V999 COMP SYNC) are displayed after conversion to DISPLAY usage SIGN LEADING.

• Nonprinting numeric values are not human-readable on the terminal display unless the CONVERSION phrase or the /DISPLAY_FORMATTED qualifier (on OpenVMS Alpha and I64) or display formatted (on Tru64 UNIX) is specified.

5. On OpenVMS, the CONVERSION phrase or /DISPLAY_FORMATTED (OpenVMS Alpha, I64), or display formatted (on Tru64 UNIX), displays floating-point items as follows:

• The floating-point data item is converted from internal floating point to E-notation representation. (E-notation consists of a mantissa and, optionally, an exponent.)

   The E-notation is described as follows:

<table>
<thead>
<tr>
<th>Size of mantissa</th>
<th>COMP-1 has 7 digits (F-floating, S-floating) plus decimal point and sign</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>COMP-2 has 15 digits (G-floating, T-floating) or 16 digits (D-floating) plus decimal point and sign</td>
</tr>
<tr>
<td>Sign of mantissa</td>
<td>Sign shown only for negative, space if positive</td>
</tr>
<tr>
<td>Form of mantissa</td>
<td>d.dddddd for COMP-1</td>
</tr>
<tr>
<td></td>
<td>d.dddddddddddddd for COMP-2 (G-floating, T-floating)</td>
</tr>
<tr>
<td></td>
<td>d.dddddddddddddd for COMP-2 (D-floating)</td>
</tr>
<tr>
<td></td>
<td>“.” replaced by “,” if DECIMAL POINT IS COMMA</td>
</tr>
<tr>
<td>Size of exponent</td>
<td>2 decimal digits for COMP-1 (F-floating, S-floating)</td>
</tr>
<tr>
<td></td>
<td>2 decimal digits for COMP-2 (D-floating)</td>
</tr>
<tr>
<td></td>
<td>3 decimal digits for COMP-2 (G-floating, T-floating)</td>
</tr>
<tr>
<td>Range of exponent</td>
<td>-38 to +38 (base 10) for COMP-1 (F-floating, S-floating)</td>
</tr>
<tr>
<td></td>
<td>-38 to +38 (base 10) for COMP-2 (D-floating)</td>
</tr>
<tr>
<td></td>
<td>-308 to +308 (base 10) for COMP-2 (G-floating, T-floating)</td>
</tr>
<tr>
<td>Form of exponent</td>
<td>E begins exponent sign shown for negative and positive; all digits shown (for example, E+01)</td>
</tr>
</tbody>
</table>

• The size of the displayed field is 13 characters for COMP-1, and 22 characters for COMP-2.

1. The CONVERSION phrase or the /DISPLAY_FORMATTED qualifier (on OpenVMS Alpha or I64) or ~display_formatted (on Tru64 UNIX) displays floating-point items as follows:

• The floating-point data item is converted from internal floating point to E-notation representation. (E-notation consists of a mantissa and, optionally, an exponent.)

• The size of the displayed field is 13 characters for COMP-1 and 22 characters for COMP-2.

• On Tru64 UNIX systems, the E-notation is described as follows:

| Size of mantissa | COMP-1 has 7 digits (plus decimal point and sign). |
### Procedure Division

<table>
<thead>
<tr>
<th>Sign of mantissa</th>
<th>COMP-2 has 16 digits (plus decimal point and sign).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form of mantissa</td>
<td>Sign shown only for negative; space if positive.</td>
</tr>
<tr>
<td>Size of exponent</td>
<td>3 decimal digits for COMP-1 and COMP-2.</td>
</tr>
<tr>
<td>Range of exponent</td>
<td>-308 to +308 (base 10) for COMP-1 and COMP-2.</td>
</tr>
<tr>
<td>Form of exponent</td>
<td>E begins exponent sign shown for negative and positive; both digits shown (for example, E+01).</td>
</tr>
</tbody>
</table>

On OpenVMS Alpha and I64 systems, compiled with the appropriate option on the /FLOAT qualifier, the E-notation is described as follows:

| Sign of mantissa | COMP-1 has 7 digits (F-floating, S-floating) plus decimal point and sign. |
|------------------| COMP-2 has 15 digits (G-floating, T-floating) or 16 digits (D-floating) plus decimal point and sign. |
| Form of mantissa | Sign shown only for negative; space if positive. |
| Size of exponent | 2 decimal digits for COMP-1 (F-floating, S-floating). |
| Size of exponent | 2 decimal digits for COMP-2 (D-floating). |
| Size of exponent | 3 decimal digits for COMP-2 (G-floating, T-floating). |
| Range of exponent| -38 to +38 (base 10) for COMP-1 (F-floating, S-floating). |
| Range of exponent| -38 to +38 (base 10) for COMP-2 (D-floating). |
| Range of exponent| -308 to +308 (base 10) for COMP-2 (G-floating, T-floating). |
| Form of exponent | E begins exponent sign shown for negative and positive; all digits shown (for example, E+01). |

### Format 1

1. The DISPLAY statement transfers data from each src-item (in its order of appearance in the statement) to output-dest.

2. No editing or conversion occurs during DISPLAY execution unless there is an applicable WITH CONVERSION phrase.

3. If src-item is a figurative constant, only one occurrence is displayed.

4. When there is more than one src-item, sending item size is the sum of the src-item sizes.
5. If there is no UPON phrase, the DISPLAY statement transfers data to the default system output device.

6. If there is no WITH NO ADVANCING phrase, the DISPLAY statement transfers device positioning information. It resets the output-dest position to the leftmost position on the next line.

7. If DECIMAL POINT IS COMMA is specified, comma replacement occurs upon display.

**Format 2**

8. The presence of either the LINE NUMBER phrase or the COLUMN NUMBER phrase implies NO ADVANCING; that is, no line feed or carriage return is generated automatically following data output. The cursor remains on the character position immediately following the position of the last character displayed. This is the default starting position for the next data item you input from or display upon the terminal.

9. If you specify neither the LINE NUMBER phrase, the COLUMN NUMBER phrase, nor the WITH NO ADVANCING phrase, data is output according to Format 1 positioning rules for the DISPLAY statement. That is, a line feed and carriage return are generated automatically following data display.

**LINE NUMBER Phrase (Format 2)**

10. The LINE NUMBER phrase positions the cursor on a specific line of the video screen prior to displaying.

11. If the LINE NUMBER phrase does not appear but the COLUMN NUMBER phrase does, then data is displayed to the current specified column position.

12. If line-num or the value of line-id is greater than the bottommost line position of the current screen, program results are undefined.

13. If you use line-id without its PLUS option, the line position is the value of line-id.

14. If you use line-id with its PLUS option, the line position is the sum of plus-num and the value of line-id.

15. If you use the PLUS option without line-id, the line position is the sum of plus-num and the value of the current line position.

16. If you use the PLUS option, but you do not specify plus-num, then PLUS 1 is implied.

17. Data output results are undefined if your program generates a value for line-id that is one of the following:
   
   - Zero
   - Negative
   - Greater than the bottommost line position of the current screen

**COLUMN NUMBER Phrase (Format 2)**

18. The COLUMN NUMBER phrase positions the cursor on a specific column of the video screen.

19. If the COLUMN NUMBER phrase does not appear but the LINE NUMBER phrase does, then data is displayed to column 1 of the specified line position.

20. If you use column-id without its PLUS option, the column position is the value of column-id.

21. If you use column-id with its PLUS option, the column position is the sum of plus-num and the value of column-id.

22. If you use the PLUS option without column-id, the column position is the sum of plus-num and the value of the current column position.
23. If you use the PLUS option, but do not specify plus-num, PLUS 1 is implied.

24. Data output results are undefined if the program generates a value for column position that is one of the following:
   - Zero
   - Negative
   - Greater than the last column position on the screen

**LINE NUMBER and COLUMN NUMBER Phrases (Format 3)**

25. The LINE NUMBER and COLUMN NUMBER phrases together give the starting screen coordinates.

26. The position of each screen item within the referenced screen-name is offset from the LINE and COLUMN positions.

27. If either LINE or COLUMN is not specified, the default value is 1.

**ERASE Phrase (Format 2)**

28. The ERASE phrase erases all, or part, of a line (or screen) before displaying data. You must specify SCREEN or LINE.

29. If you use its TO END option, the ERASE phrase erases the line (or screen) from the implied, or stated, cursor position to the end of the line (or screen).

30. If you do not use its TO END option, the ERASE phrase erases the entire line (or screen).

**BELL Phrase (Format 2)**

31. The BELL phrase rings the terminal bell before displaying data.

**UNDERLINED Phrase (Format 2)**

32. The UNDERLINED phrase displays characters on the screen with the underscore on character attribute.

**BOLD Phrase (Format 2)**

33. The BOLD phrase displays characters on the screen with the bold on character attribute. The BOLD attribute is only detectable when any of the following conditions are true:
   - Nonspace characters are displayed.
   - The underlined or reversed attributes are specified.
   - The terminal screen is set to light background.

**BLINKING Phrase (Format 2)**

34. The BLINKING phrase displays characters on the screen with the blink on character attribute. The BLINKING attribute is only detectable when any of the following conditions are true:
   - Nonspace characters are displayed.
   - The underlined or reversed attributes are specified.
The terminal screen is set to light background.

**REVERSED Phrase (Format 2)**

1. The REVERSED phrase displays characters on the screen with the *reverse video on* character attribute.

**Formats 4, 5, and 6**

1. When a Format 4 DISPLAY statement is specified, the value stored in `arg-position` is moved to `argument-number`. This updates the current argument position indicator for the command line (see ARGUMENT-NUMBER in the SPECIAL-NAMES paragraph in Chapter 4, Environment Division). This points to the selected argument to be read by a Format 7 ACCEPT statement.

2. `arg-position` must be in the range 0 to 99 and can refer to arguments, switches, and flags that appear on the run command line of the COBOL program. When the current argument position indicator is zero, it refers to the zeroth command line argument, in other words the command that invoked the COBOL program.

3. When a Format 5 DISPLAY statement is specified, the value stored in `envlog-name` is moved to `environment-name` (see ENVIRONMENT-NAME in the SPECIAL-NAMES paragraph in Chapter 4, Environment Division). The updated value of `environment-name` becomes the environment variable or logical to be accessed by subsequent Format 6 DISPLAY and Format 8 ACCEPT statements.

4. `environment-value`, when used with a Format 6 DISPLAY, receives the value stored in `envlog-value`. The environment variable or logical is the one named by a Format 5 DISPLAY statement (see ENVIRONMENT-VALUE in the SPECIAL-NAMES paragraph in Chapter 4, Environment Division).

5. `stmt` is executed if the name of the environment variable or logical has not been set by a Format 5 DISPLAY, or if the environment variable or logical does not exist.

6. `stmt2` is executed if the exception condition does not exist.

**Technical Notes**

**Format 1**

1. On OpenVMS, the DISPLAY statement transfers data through the I/O system (RMS), using the Variable with Fixed-Length Control (VFC) format.

2. A DISPLAY statement without the UPON phrase transfers data to the default output device (the terminal). To transfer data to a file on Tru64 UNIX systems, the environment variable COBOL_OUTPUT can be used to specify a text file containing output data. To transfer data to a file on OpenVMS systems the logical COB SOUTPUT or SYSS$OUTPUT can be used to specify a text file containing output data.

Alternatively, output device redirection (`>`) can be used on Tru64 UNIX systems to name an output file.

3. A DISPLAY statement that includes the UPON phrase transfers data to the `file-device-name` associated with the SPECIAL-NAMES paragraph description of `output-dest`.

4. Because the object of a logical name (on OpenVMS systems) is not necessarily a device, no open mode is implied. As a result, `output-dest` can be associated with any `device-name` in the SPECIAL-NAMES paragraph. For example, `output-dest` can refer to PAPER-TAPE-READER as well as PAPER-TAPE-PUNCH.

**Format 2**

5. Format 2 is a VSI extension to the standard COBOL use of the DISPLAY statement.

6. The VSI extensions to the ACCEPT and DISPLAY statements support data input and display only on the VT100 and later terminal types, including emulators of these terminal types.
7. The UNDERLINED, BOLD, BLINKING, and REVERSED character attributes are not available on VT100 terminals without the advanced video option.

8. You should display data only on fields that are within screen boundaries. That is, the terminal operator should see all the characters displayed. If data is displayed on fields that position the cursor outside screen boundaries, it does not result in an error condition. However, your program might not produce the results you expect.

Values for screen boundaries depend on the terminal type and the column mode in which it is operating. Refer to the appropriate terminal user's guide for more information on screen boundaries.

9. Line positioning can be a one- or two-step process. The first (or only) step is absolute positioning, which is using the value of line-num or line-id to determine the line position. The second step is relative positioning, which is adding the value of plus-num to line-id to determine the line position.

The following sample statements would produce undefined results because they use absolute line positioning to reach a line beyond the bottom of the screen (assume ITEMB has a value of 25):

DISPLAY SRC-EXAMPLE AT LINE NUMBER 25.
DISPLAY SRC-EXAMPLE AT LINE NUMBER ITEMB.
DISPLAY SRC-EXAMPLE AT LINE NUMBER ITEMB PLUS 0.

The last DISPLAY statement illustrates that use of the PLUS option does not necessarily mean that relative positioning will always occur. When you specify line-id, absolute line positioning always occurs before a PLUS option can execute. In this case, line-id (ITEMB) is specified, and it has a value of 25. Therefore, the line position is outside the screen boundary before the PLUS option executes, and program results are undefined.

10. When there is more than one src-item, each specific src-item is displayed, after application of any phrases specific to that src-item, in order of occurrence in the DISPLAY statement.

Formats 2 and 3

11. On OpenVMS, control sequences from SMGTERMS.TXT are used to accomplish cursor positioning, screen erasure, and video attributes. Refer to the Support for Non-HP Terminals chapter of the OpenVMS RTL Screen Management (SMGS) Manual if you wish to customize SMGTERMS.TXT.

All Formats

12. VSI COBOL parses the contents of the data being displayed to determine how they affect the terminal and the cursor position. The parsing of control sequences is performed according to the DEC STD 138-0 Registry of Control Functions for Character Imaging Devices. VSI COBOL does not modify the control sequences in any way; if an invalid control sequence is found, VSI COBOL does not attempt to correct the sequence.

Therefore when you display an escape or control sequence, the entire sequence must be displayed in one operation:

• If you use DISPLAY Format 1, the complete sequence must be contained in one or more src-items within one DISPLAY statement.
• If you use DISPLAY Format 2, the complete sequence must be contained in one src-item.
• If you use DISPLAY Format 3, the complete sequence must be contained within one elementary screen item.

13. A DISPLAY statement used in an ACCEPT [NOT] ON EXCEPTION statement must be terminated (with, for example END-DISPLAY) on Alpha and I64 systems. If you are concerned with the different VAX behavior, refer to the appendix on compatibility in the VSI COBOL User Manual.

Additional References

• SPECIAL-NAMES section in Chapter 4, Environment Division
• VSI COBOL User Manual sections on using command line arguments and environment variables

Examples

In the example results, the character “s” represents a space. The examples assume a maximum screen size of 24 lines. They also assume the following Environment and Data Division entries:

SPECIAL-NAMES.
   LINE-PRINTER IS ERR-REPORTER.

01 ITEMA PIC X(6) VALUE "ITEMS ".
01 ITEMB PIC X(8) VALUE "VALID ".
01 ITEMC PIC X(5) VALUE "TODAY ".
01 ITEMD PIC 99 VALUE 2.
01 ITEME PIC X(10) VALUE "MONDAY ".

RESULT:

1. DISPLAY ITEMC.
   TODAY
2. DISPLAY ITEMD UPON ERR-REPORTER.
   02
3. DISPLAY ITEMD ITEMA "ARE" ITEMB.
   02ITEMSsAREVALIDss
4. DISPLAY ITEMD SPACE ITEMA "AREs" ITEMB.
   02sITEMSsAREsVALIDss
5. DISPLAY ITEMC "sISs" NO ADVANCING.
   DISPLAY ITEME.
   DISPLAY ITEME.
   TODAYsISsMONDAYssss
   MONDAYssss

The following program uses VSI DISPLAY extensions (Format 2).

IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLES.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ITEMF        COMP-1.
01 ITEMG        COMP-2.
01 ITEMH        PIC S9(9) COMP VALUE IS 123456789.
01 ITEMI        PIC S9(9) COMP-3.
PROCEDURE DIVISION.
01.
   MOVE 101.000000000 TO ITEMF.
   MOVE .109999999 TO ITEMG.
   MOVE 123456789 TO ITEMI.
   DISPLAY
      ITEMF WITH CONVERSION LINE PLUS
      ITEMG WITH CONVERSION LINE PLUS
      ITEMH WITH CONVERSION LINE PLUS
      ITEMI WITH CONVERSION LINE PLUS
      .
      .

The VSI COBOL User Manual contains additional examples using VSI extensions to the DISPLAY statement. Refer to the chapters that describe screen handling, command line variables, environment variables and logicals.
DIVIDE

DIVIDE

Function

The DIVIDE statement divides one or more numeric data items by another and sets the value of the data items equal to the quotient, optionally storing the remainder.

[num]

is a numeric literal or the identifier of an elementary numeric item.

[result]

is the identifier of an elementary numeric item or an elementary numeric edited item. However, in Format 1, result must be an elementary numeric item. It is the resultant identifier.

[stmt]

is an imperative statement executed when a size error condition has occurred.

[stmt2]

is an imperative statement executed when no size error condition has occurred.

[remaind]

is the identifier of an elementary numeric item or an elementary numeric edited item.

General Rules

Format 1

1. The value of num is divided into the value of the first result. This quotient replaces the current value of the first result. The process repeats for each of the other occurrences of result.

Format 2

2. The value of the first num is divided into the value of the second. This quotient replaces the current value of each result.

Format 3

3. The value of the first num is divided by the value of the second. This quotient replaces the current value of each result.

Formats 4 and 5

4. These formats produce a remainder (remaind) from the division operation. The remainder is the result of subtracting the product of the quotient (result) and the divisor from the dividend.

If result refers to a numeric edited item, the quotient is an equivalent unedited intermediate field. For example, if you describe result with the PICTURE -ZZ.99, the compiler uses an intermediate field with the implicit PICTURE S99V99.
When the ROUNDED phrase is present, the remainder computation uses an intermediate quotient field that is truncated rather than rounded.

5. The computation described in rule 4 determines the accuracy of remaind. It includes decimal point alignment and truncation (not rounding) required by the description of remaind.

6. When the ON SIZE ERROR phrase is present:
   • If the size error occurs on result, the contents of both result and remaind are unchanged.
   • If the size error occurs on remaind, its contents are unchanged.

Additional References

• Section 6.1.4: Scope of Statements
• Section 6.6.1: Arithmetic Operations
• Section 6.6.2: Multiple Receiving Fields in Arithmetic Statements
• Section 6.6.3: ROUNDED Phrase
• Section 6.6.4: ON SIZE ERROR Phrase
• Section 6.6.7: Overlapping Operands and Incompatible Data

Examples

The following example shows a run-time message issued for an illegal attempt to divide by zero:

%COB-E-DIVBY-ZER, divide by zero; execution continues

Each of the examples assume the following data descriptions and initial values. The initial values are listed in the righthand column:

INITIAL VALUES

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEM</td>
<td>PIC 99V99</td>
<td>VALUE 9</td>
</tr>
<tr>
<td>ITEMB</td>
<td>PIC 99V99</td>
<td>VALUE 24</td>
</tr>
<tr>
<td>ITEMC</td>
<td>PIC 99V99</td>
<td>VALUE 8</td>
</tr>
<tr>
<td>ITEMD</td>
<td>PIC 99</td>
<td>VALUE 12</td>
</tr>
<tr>
<td>ITEME</td>
<td>PIC 99V99</td>
<td>VALUE 3</td>
</tr>
<tr>
<td>ITEMF</td>
<td>PIC 99</td>
<td>VALUE 47</td>
</tr>
<tr>
<td>ITEMG</td>
<td>PIC 9</td>
<td>VALUE 9</td>
</tr>
<tr>
<td>ITEMH</td>
<td>PIC 9</td>
<td>VALUE 2</td>
</tr>
<tr>
<td>ITEMI</td>
<td>PIC 99</td>
<td>VALUE 4</td>
</tr>
</tbody>
</table>

In each of the following examples, the righthand column shows the results of the DIVIDE operation.

1. Without GIVING phrase or rounding:

   RESULTS
   
   DIVIDE ITEM INTO ITEMB.  ITEMB = 2.66

2. With rounding:

   DIVIDE ITEM INTO ITEMB ROUNDED.  ITEMB = 2.67

3. GIVING phrase:
DIVIDE ITEMA INTO ITEMB GIVING ITEMD.  ITEMD = 2

4. GIVING phrase with rounding:
DIVIDE ITEMA INTO ITEMB GIVING ITEMD ROUNDED.  ITEMD = 3

5. BY phrase:
DIVIDE ITEMA BY ITEMB  GIVING ITEMD.  ITEMD = 0

6. REMAINDER phrase:
DIVIDE ITEMA INTO ITEMB GIVING ITEMD REMAINDER ITEMC.  ITEMD = 2  ITEMC = 6.00

7. REMAINDER phrase with rounding:
DIVIDE ITEMA INTO ITEMB GIVING ITEMD ROUNDED REMAINDER ITEMC.  ITEMD = 3  ITEMC = 6.00

8. Effects of decimal alignment on quotient and remainder:
DIVIDE ITEMA INTO ITEMB GIVING ITEME REMAINDER ITEMC.  ITEME = 2.66  ITEMC = .06

9. Effects of decimal alignment on remainder and quotient with rounding:
DIVIDE ITEMA INTO ITEMB GIVING ITEME ROUNDED REMAINDER ITEMC.  ITEME = 2.67  ITEMC = .06

10. The ON SIZE ERROR phrase: (IF ON SIZE ERROR occurs on an occurrence of result, the contents of that occurrence of result are unchanged.)
DIVIDE ITEME INTO ITEMF GIVING ITEMG ITEMD  ON SIZE ERROR MOVE 0 TO ITEMH.  ITEMD = 15  ITEMG = 9  ITEMH = 0

11. The ON SIZE ERROR phrase:
(IF ON SIZE ERROR occurs on remainder, the contents of remainder are unchanged.)
DIVIDE ITEMD INTO ITEMF GIVING ITEMI REMAINDER ITEMG  ON SIZE ERROR MOVE 0 TO ITEMH.  ITEMI = 3  ITEMG = 9  ITEMH = 0

12. The NOT ON SIZE ERROR phrase:
DIVIDE ITEMD INTO ITEMF GIVING ITEMI REMAINDER ITEMC  ON SIZE ERROR MOVE 0 TO ITEMH NOT ON SIZE ERROR MOVE 1 TO ITEMH.  ITEMI = 3  ITEMC = 11.00  ITEMH = 1
EVALUATE Function

The EVALUATE statement selects a program action based on the evaluation of one or more conditions.

General Format

[subj-item]

is an identifier, an arithmetic or conditional expression, or a literal other than the figurative constant ZERO.

[cond]

is a conditional expression.

[obj-item]

is a literal, an identifier, or an arithmetic expression.

[stmt1]

is an imperative statement.

[stmt2]

is an imperative statement.

Syntax Rules

1. Before the first WHEN phrase: (a) subj-item and the words TRUE and FALSE are called subjects, and (b) all subjects comprise the subject set.

2. In a WHEN phrase: (a) ANY, TRUE, FALSE, and the operands are called objects, and (b) all objects in a single WHEN phrase comprise an object set.

3. The number of objects in the object set must equal the number of subjects in the subject set.

4. The words THROUGH and THRU are equivalent.

5. Two obj-items connected by a THROUGH phrase:
   - Must be of the same class
   - Combine to form one object

6. Each object in an object set must correspond to the subject by appearing in the same ordinal position as in the subject set. For each pair:
   - The obj-item must be a valid operand for comparison to the subject.
   - TRUE, FALSE, or cond as an object must correspond only to TRUE, FALSE, or a conditional expression as the subject.
   - ANY can correspond to any type of subject.

7. Conditional expressions can be simple or complex conditions.
General Rules

Evaluation Procedure

1. The EVALUATE statement operates as if each subject and object were evaluated and assigned one of the following:
   - A numeric or nonnumeric value
   - A range of numeric or nonnumeric values
   - A truth value

   The statement assigns values according to the following rules:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Value Assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. An identifier for a subject, or for an object without the NOT or THROUGH phrases</td>
<td>Value and class of the identifier's data item.</td>
</tr>
<tr>
<td>b. A literal for a subject, or for an object without the NOT or THROUGH phrases</td>
<td>Value and class of the literal.</td>
</tr>
<tr>
<td>c. The figurative constant ZERO for an object without the NOT or THROUGH phrases</td>
<td>Value and class of the corresponding subject.</td>
</tr>
<tr>
<td>d. An arithmetic expression for a subject, or for an object without the NOT or THROUGH phrases</td>
<td>Numeric value, according to the rules for evaluating arithmetic expressions.</td>
</tr>
<tr>
<td>e. A conditional expression for a subject or a conditional expression for an object</td>
<td>Truth value, according to the rules for evaluating conditional expressions.</td>
</tr>
<tr>
<td>f. TRUE or FALSE as a subject or object</td>
<td>Truth value: true for the word TRUE and false for the word FALSE.</td>
</tr>
<tr>
<td>g. ANY for an object</td>
<td>No further evaluation.</td>
</tr>
<tr>
<td>h. THROUGH phrase for an object without the NOT phrase</td>
<td>The range of values is all values that, when compared to the subject, are greater than or equal to the first obj-item and less than or equal to the second obj-item. If the first obj-item is greater than the second obj-item, there are no values in the range.</td>
</tr>
<tr>
<td>i. Object with the NOT phrase</td>
<td>All values not equal to the value (or range of values) that would be assigned without the NOT phrase.</td>
</tr>
</tbody>
</table>

Comparison Procedure

1. After values have been assigned to each subject and object, comparison begins. It proceeds as if the values were compared to determine if any WHEN phrase satisfies the subject set.

2. EVALUATE compares each object in the object set of the first WHEN phrase to the subject in the same ordinal position in the subject set. The comparison is satisfied if one of the following conditions is true:
   - The items being compared are assigned numeric or nonnumeric values or a range of numeric or nonnumeric values; and the value assigned to the subject equals the value, or one of the range of values, assigned to the object, according to the rules for comparison.
   - The items being compared are assigned identical truth values.
   - The word ANY represents the object.
3. If the comparison is satisfied for every object in an object set, the WHEN phrase containing that object set is selected.

4. If the comparison is not satisfied for every object in an object set, the object set does not satisfy the subject set.

5. The comparison procedure is repeated for each object set, in order of appearance, until one of these conditions occur:
   • A WHEN phrase is selected by satisfying the subject set.
   • A WHEN OTHER phrase is selected.
   • There are no more object sets.
   • The END-EVALUATE statement is reached.
   • A separator period is reached.

**Execution Procedure**

1. If a WHEN phrase is selected, execution continues with `{stmt1}`.

2. If no WHEN phrase is selected, and a WHEN OTHER phrase is present, execution continues with `{stmt2}`.

3. EVALUATE statement execution ends when one of the following conditions occurs:
   • Execution reaches the end of the selected WHEN phrase.
   • Execution reaches the end of the WHEN OTHER phrase.
   • No WHEN phrase is selected and there is no WHEN OTHER phrase.
   • Execution reaches END-EVALUATE.
   • Execution reaches a separator period.

**Additional References**

• Section 6.1.4: Scope of Statements
• Section 6.4: Arithmetic Expressions
• Section 6.5: Conditional Expressions
• Section 6.5.1: Relation Conditions
• Section 6.6.1: Arithmetic Operations

**Examples**

In these examples, the results are shown as either data item values or procedure branches. However, `{stmt}` can be any imperative statement, including multiple statements.

1. One condition.

   ```plaintext
   EVALUATE ITEM A
   WHEN "A01" MOVE 1 TO ITEMB
   WHEN "A02" THRU "C16" MOVE 2 TO ITEMB
   WHEN "C20" THRU "L86" MOVE 3 TO ITEMB
   WHEN "R20" ADD 1 TO R-TOT
   ```
Procedure Division

GO TO PROC-A
WHEN OTHER MOVE 0 TO ITEMB
END-EVALUATE.

Samples:

<table>
<thead>
<tr>
<th>ITEMA</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;A15&quot;</td>
<td>ITEMB = 2</td>
</tr>
<tr>
<td>&quot;P80&quot;</td>
<td>ITEMB = 0</td>
</tr>
<tr>
<td>&quot;F01&quot;</td>
<td>ITEMB = 3</td>
</tr>
<tr>
<td>&quot;M19&quot;</td>
<td>ITEMB = 0</td>
</tr>
<tr>
<td>&quot;A01&quot;</td>
<td>ITEMB = 1</td>
</tr>
<tr>
<td>&quot;R20&quot;</td>
<td>PROC-A</td>
</tr>
</tbody>
</table>

2. Multiple conditions. This example shows how EVALUATE can represent a decision table.

EVALUATE LOW-STOK WEEK-USE LOC-VNDR ON-ORDER
WHEN "Y", 16 THRU 999, ANY, "N" GO TO RUSH-ORDER
WHEN "Y", 16 THRU 999, ANY, "Y" GO TO NORMAL-ORDER
WHEN "Y", 8 THRU 15, "N", "N" GO TO RUSH-ORDER
WHEN "Y", 8 THRU 15, "Y", "N" GO TO NORMAL-ORDER
WHEN "Y", 0 THRU 7, ANY, "N" GO TO NORMAL-ORDER
WHEN "N", ANY, ANY, "Y" GO TO CANCEL-ORDER
END-EVALUATE.

Samples:

<table>
<thead>
<tr>
<th>LOW-STOK</th>
<th>WEEK-USE</th>
<th>LOC-VNDR</th>
<th>ON-ORDER</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Y&quot;</td>
<td>38</td>
<td>&quot;N&quot;</td>
<td>&quot;Y&quot;</td>
<td>NORMAL-ORDER</td>
</tr>
<tr>
<td>&quot;N&quot;</td>
<td>20</td>
<td>&quot;Y&quot;</td>
<td>&quot;Y&quot;</td>
<td>CANCEL-ORDER</td>
</tr>
<tr>
<td>&quot;N&quot;</td>
<td>12</td>
<td>&quot;Y&quot;</td>
<td>&quot;N&quot;</td>
<td>next statement</td>
</tr>
<tr>
<td>&quot;Y&quot;</td>
<td>12</td>
<td>&quot;Y&quot;</td>
<td>&quot;N&quot;</td>
<td>NORMAL-ORDER</td>
</tr>
<tr>
<td>&quot;Y&quot;</td>
<td>12</td>
<td>&quot;Y&quot;</td>
<td>&quot;Y&quot;</td>
<td>next statement</td>
</tr>
<tr>
<td>&quot;Y&quot;</td>
<td>40</td>
<td>&quot;N&quot;</td>
<td>&quot;N&quot;</td>
<td>RUSH-ORDER</td>
</tr>
</tbody>
</table>

3. Relation conditions and arithmetic expressions.

EVALUATE-ITEM-Routine.

* After the imperative statement in the selected WHEN phrase executes (for example PERFORM PROC-A), control then transfers to the first statement following the end of the EVALUATE statement (MOVE A TO B).

* EVALUATE ITEM > 6 AND < 30, 8 * ITEMB - 1
WHEN TRUE, 5 * ITEMC PERFORM PROC-A
WHEN FALSE, ITEMC PERFORM PROC-B
WHEN ITEMC > 12, -1 PERFORM PROC-C
WHEN TRUE, NOT 7 THRU 40 PERFORM PROC-D
WHEN OTHER PERFORM PROC-E
END-EVALUATE.

MOVE A TO B.
Samples:

<table>
<thead>
<tr>
<th>ITEMA</th>
<th>ITEMB</th>
<th>ITEMC</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>2</td>
<td>3</td>
<td>PROC-A</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>14</td>
<td>PROC-C</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>14</td>
<td>PROC-E</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>23</td>
<td>PROC-B</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>5</td>
<td>PROC-D</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>11</td>
<td>PROC-C</td>
</tr>
</tbody>
</table>

Consider how the EVALUATE statement works using the values in the previous sample:

1. The value of the first subject is a truth value (General Rule 1e). ITEMA is *not* greater than 6 and less than 30; therefore, the value of the first subject is false.

2. The value of the second subject is a numeric value (General Rule 1d):
   \[8 \times 0 - 1 = -1.\]

3. When the first WHEN phrase is evaluated:
   - The value of the first object is a truth value (General Rule 1f): true.
   - The value of the second object is a numeric value: 55.
   - The value of the first object does not equal that of the first subject. Furthermore, the values of the second object and subject do not match. Therefore, this WHEN phrase is not selected (General Rule 5).

4. When the second WHEN phrase is evaluated:
   - The value of the first object is a truth value (General Rule 1f): false.
   - The value of the second object is a numeric value: 11.
   - The value of the first object equals that of the first subject. However, the values of the second object and subject do not match. Therefore, this WHEN phrase is not selected (General Rule 5).

5. When the third WHEN phrase is evaluated:
   - The value of the first object is a truth value (General Rule 1f). Because the value of ITEMC is *not* greater than 12, the value of this object is false.
   - The value of the second object is a numeric value: -1.
   - The value of the first object equals that of the first subject. The values of the second object and subject also match. Therefore, this WHEN phrase is selected (General Rule 4).

6. The statement following the third WHEN phrase is PERFORM PROC-C. Control transfers to that procedure, and the EVALUATE statement ends.

**EXIT**

**Function**

The EXIT statement provides a common logical end point for a series of procedures.
Syntax Rule

The EXIT statement must appear in a sentence by itself and be the only sentence in the paragraph.

General Rule

The EXIT statement associates a procedure-name with a point in the program. It has no other effect on program compilation or execution.

Example

REPORT-INVALID-ADD.
   DISPLAY " ".
   DISPLAY "INVALID ADDITION".
   DISPLAY "RECORD ALREADY EXISTS".
   DISPLAY "UPDATE ATTEMPT: " UPDATE-REC.
   DISPLAY "EXISTING RECORD: " OLD-REC.
REPORT-INVALID-ADD-EXIT.
EXIT.

EXIT PROGRAM

EXIT PROGRAM

Function

The EXIT PROGRAM statement marks the logical end of a called program.

Syntax Rules

1. If the EXIT PROGRAM statement is in a consecutive sequence of imperative statements, it must be the last statement in that sequence.

2. The EXIT PROGRAM statement cannot appear in a GLOBAL USE procedure.

General Rules

1. If EXIT PROGRAM executes in a program that is not a called program, it causes execution to continue with the next executable statement. Refer to the VSI COBOL User Manual for information on how the \texttt{v3} setting of the standard compiler option affects the EXIT PROGRAM statement.

2. If the EXIT PROGRAM statement executes in a called program without the \texttt{INITIAL} clause in its PROGRAM-ID paragraph, execution continues with the next executable statement after the \texttt{CALL} statement in the calling program.

   The state of the calling program does not change; it is the same as when the program executed the \texttt{CALL} statement. However, the contents of data items and the positioning of data files shared by the calling and called programs can change.

   The state of the called program does not change. However, the called program is considered to have reached the ends of the ranges of all \texttt{PERFORM} statements it executed. Therefore, an error does not occur if the called program is entered again during image execution.

3. When EXIT PROGRAM executes in a called program with the \texttt{INITIAL} attribute, the actions described in General Rule 2 also apply. In addition, executing the EXIT PROGRAM statement is equivalent to executing a CANCEL statement that names the called program.
4. Special handling of the EXIT PROGRAM statement is performed when you specify the standard compiler option with the v3 setting on the compiler command line. Refer to the VSI COBOL User Manual for more information.

**Example**

TEST-RETURN.
  IF ITEMA NOT = ITEMB
    MOVE ITEMA TO ITEMB
  EXIT PROGRAM.

**GENERATE**

**Function**

The GENERATE statement directs the Report Writer Control System (RWCS) to produce a report according to the Report Description entry (RD) in the Report Section of the Data Division.

[report-item]

names either a report-name in a Report Description entry, or the group-data-name of a TYPE IS DETAIL report group.

**Syntax Rules**

1. If *report-item* references a group-data-name, it must name a TYPE DETAIL report group. Group-data-name can be qualified by report-name.

2. If *report-item* references a report-name, its report description must contain:
   
   • A CONTROL clause
   
   • At least one CONTROL HEADING, DETAIL, or CONTROL FOOTING report group
   
   • No more than one DETAIL report group

**General Rules**

1. An INITIATE statement must be executed before a GENERATE statement is executed for a specific report.

2. The RWCS produces a summary report if all of the GENERATE statements for a report reference report-name. A summary report contains no TYPE IS DETAIL report groups.

3. The RWCS produces a detail report if a GENERATE statement references a DETAIL report group.

4. To detect and trigger control breaks for a specific report, the RWCS:
   
   • Saves the initial values within control data items as prior values when the GENERATE statement executes.
   
   • Compares the prior values to the current values of control data items when subsequent GENERATE statements execute. Only if the current values change does a control break occur. If a control break occurs, the current values are saved as prior values.
   
   • Repeats the preceding step until the last control break is processed.
5. The RWCS automatically processes any PAGE HEADING and PAGE FOOTING report groups when it must start a new page to present a CONTROL HEADING, DETAIL, or CONTROL FOOTING.

6. When the first GENERATE statement for a specific report is executed, the RWCS processes these report groups, if present in the report description, in this order:
   a. The REPORT HEADING report group.
   b. The PAGE HEADING report group.
   c. All CONTROL HEADING report groups from major to minor.
   d. For GENERATE group-data-name statements (detail reporting), the RWCS presents the specific DETAIL report group for processing.
   e. For GENERATE report-name statements (summary reporting), the RWCS does not present the DETAIL report group for processing; however, the RWCS does perform all other DETAIL report group functions.

7. When subsequent GENERATE statements are executed for a specific report, the RWCS:
   • Checks for control breaks. The rules governing the inequality of control data items are identical to the rules for relation conditions. If a control break occurs, the RWCS:
     a. Enables the CONTROL FOOTING USE procedures and CONTROL FOOTING SOURCE clauses. This allows program access to the control data item values that the RWCS uses to detect a given control break.
     b. Processes the CONTROL FOOTING report groups starting with the minor. Only CONTROL FOOTING report groups less major than the highest level at which a control break occurs are processed.
     c. Processes the CONTROL HEADING report groups in the order major to minor. Only the CONTROL HEADING report groups less major than the highest level at which a control break occurs are processed.
   • Processes the GENERATE statement. For GENERATE group-data-name statements (detail reporting), the RWCS processes the specific DETAIL report group. For GENERATE report-name statements (summary reporting), the RWCS does not present the DETAIL report group for processing; however, the RWCS does perform all other DETAIL report group functions.

8. No GENERATE statements can reference a file after executing a TERMINATE statement for the same file.

**Additional References**

• TYPE clause in Chapter 5, *Data Division*

• USE statement

**GO TO**

**Function**

The GO TO statement transfers control from one part of the Procedure Division to another.

[proc-name]

is a procedure-name.

[num]

is the identifier of an elementary numeric item described with no positions to the right of the assumed decimal point.
Syntax Rules

1. A Format 1 GO TO statement that is in a consecutive sequence of imperative statements in a sentence must be the last statement in the sentence.

2. If an ALTER statement refers to a paragraph, the paragraph must consist of only a paragraph header followed by a Format 1 GO TO statement.

3. A Format 1 GO TO statement without proc-name can only be in a single-statement paragraph.

General Rules

Format 1

1. The GO TO statement transfers control to proc-name.

2. If there is no proc-name, the GO TO statement cannot execute before an ALTER statement changes its destination.

Format 2

1. The GO TO statement transfers control to the proc-name in the ordinal position indicated by the value of num.

   No transfer occurs, and control passes to the next executable statement if the value of num is one of the following:
   
   • Not greater than zero

   • Greater than the number of proc-names in the statement

Examples

1. Format 1:
   
   GO TO ENDING-ROUTINE.

2. Format 2:
   
   GO TO FRESHMAN
   SOPHOMORE
   JUNIOR
   SENIOR
   DEPENDING ON YEAR-LEVEL.

   MOVE ...

<table>
<thead>
<tr>
<th>YEAR-LEVEL</th>
<th>Transfers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FRESHMAN label</td>
</tr>
<tr>
<td>2</td>
<td>SOPHOMORE label</td>
</tr>
<tr>
<td>3</td>
<td>JUNIOR label</td>
</tr>
<tr>
<td>4</td>
<td>SENIOR label</td>
</tr>
<tr>
<td>5</td>
<td>MOVE statement</td>
</tr>
<tr>
<td>0</td>
<td>MOVE statement</td>
</tr>
<tr>
<td>-10</td>
<td>MOVE statement</td>
</tr>
</tbody>
</table>
IF

Function

The IF statement evaluates a condition. The condition's truth value determines the program action that follows.

[statement-1]

is an imperative or conditional statement. An imperative statement can precede a conditional statement.

[statement-2]

is an imperative or conditional statement. An imperative statement can precede a conditional statement.

Syntax Rules

1. The ELSE NEXT SENTENCE phrase is optional if it immediately precedes a separator period.

2. If the END-IF phrase is specified, the NEXT SENTENCE phrase must not be specified.

General Rules

1. The scope of an IF statement ends with any of the following:
   • An END-IF phrase at the same nesting level
   • A separator period
   • An ELSE phrase associated with an IF statement at a higher nesting level

2. If the condition is true, the following control transfers occur:
   • If there is a statement-1, it executes.
     
     statement-1 can contain a procedure branching or conditional statement. Control then transfers according to the rules of the statement.
     
     Otherwise, the ELSE phrase (if any) is ignored. Control passes to the end of the IF statement.
   • If you use NEXT SENTENCE instead of statement-1, the ELSE phrase (if any) is ignored. Control passes to the next executable sentence.

3. If the condition is false, the following control transfers occur:
   • statement-1 or its substitute NEXT SENTENCE is ignored. If statement-2 is used, it executes.
     
     statement-2 can contain a procedure branching or conditional statement. Control then transfers according to the rules of the statement. Otherwise, control passes to the end of the IF statement.
     
     • If there is no ELSE phrase, statement-1 is ignored. Control passes to the end of the IF statement.
   • If the ELSE NEXT SENTENCE phrase is present, statement-1 is ignored. Control passes to the next executable sentence.

4. An IF statement can appear in either or both statement-1 and statement-2. In this case, the IF statement is considered nested, because its scope is entirely within the scope of another IF statement.

5. IF statements within IF statements are paired combinations, beginning with IF and ending with ELSE or END-IF; this pairing proceeds from left to right. Thus, an ELSE or END-IF phrase applies to the first preceding unpaired IF.
Additional References

- Section 6.1: Verbs, Statements, and Sentences
- Section 6.1.4: Scope of Statements
- Section 6.5: Conditional Expressions

Examples

1. No ELSE phrase:

   IF ITEMA < 20
      MOVE "X" TO ITEMB.

<table>
<thead>
<tr>
<th>ITEMA</th>
<th>ITEMB</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>&quot;X&quot;</td>
</tr>
<tr>
<td>35</td>
<td>?</td>
</tr>
<tr>
<td>19</td>
<td>&quot;X&quot;</td>
</tr>
</tbody>
</table>

2. With ELSE phrase:

   IF ITEMA > 10
      MOVE "X" TO ITEMB
   ELSE
      GO TO PROC-A.
      ADD ...

<table>
<thead>
<tr>
<th>ITEMA</th>
<th>Next Statement</th>
<th>ITEMB</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>ADD</td>
<td>&quot;X&quot;</td>
</tr>
<tr>
<td>8</td>
<td>PROC-A</td>
<td>?</td>
</tr>
</tbody>
</table>

3. With NEXT SENTENCE phrase:

   (In each case, the next executable statement is the ADD statement.)

   IF ITEMA < 10 OR > 20
      NEXT SENTENCE
   ELSE
      MOVE "X" TO ITEMB.
      ADD ...

<table>
<thead>
<tr>
<th>ITEMA</th>
<th>ITEMB</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>?</td>
</tr>
<tr>
<td>17</td>
<td>&quot;X&quot;</td>
</tr>
<tr>
<td>35</td>
<td>?</td>
</tr>
</tbody>
</table>

4. Nested IF statements:

   IF ITEMA > 10
      IF ITEMA = ITEMC
         MOVE "X" TO ITEMB
      ELSE
         MOVE "Y" TO ITEMB
      ELSE
         GO TO PROC-A.
ADD ...

<table>
<thead>
<tr>
<th>Input Values</th>
<th>Output Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEMA</td>
<td>ITEMC</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

5. END-IF:

(In this example, the initial value of ITEMD is 5.)

IF ITEMA > 10
    IF ITEMA = ITEMC
        ADD 1 TO ITEMD
        MOVE "X" TO ITEMB
    END-IF
    ADD 1 TO ITEMD.

<table>
<thead>
<tr>
<th>ITEMA</th>
<th>ITEMC</th>
<th>ITEMB</th>
<th>ITEMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
<td>?</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>?</td>
<td>6</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>“X”</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>?</td>
<td>5</td>
</tr>
</tbody>
</table>

### INITIALIZE

#### INITIALIZE

#### Function
The INITIALIZE statement sets selected types of data fields to predetermined values.

#### General Format

[fld-name]

is the identifier of the receiving area data item.

[val]

is the sending area. It can be a literal or the identifier of a data item.

#### Syntax Rules

1. The phrase after the word REPLACING is the category phrase.

2. The category of the data item referred to by val must be consistent with that in the category phrase. The combination of categories must allow execution of a valid MOVE statement.

3. The same category cannot be repeated in a REPLACING phrase.

4. The description of fld-name or any item subordinate to it cannot contain the OCCURS clause DEPENDING phrase.
5. Neither fld-name nor val can be index data items.

6. fld-name cannot contain a RENAMES clause.

**General Rules**

1. The key word that follows the word REPLACING corresponds to a category of data. (See the section on Categories and Classes of Data in the Data Division chapter.)

2. fld-name can be an elementary or group item. If it is a group item, the INITIALIZE statement operates on the elementary items within the group item. For a table within a group item, INITIALIZE operates on the elementary items within the table.

3. Whether fld-name is an elementary item or a group item, if the REPLACING phrase is specified, all data movement operations occur as if they resulted from a series of MOVE statements with elementary item receiving areas:
   - If the receiving area is a group item, INITIALIZE affects only those subordinate elementary items whose category matches a category phrase. General Rule 6 describes the effect on elementary items when there is no REPLACING phrase.
   - INITIALIZE affects all eligible elementary items, including all occurrences of table items in the group.
   - If the receiving area is an elementary item, that item is initialized only if it matches a category phrase.

4. INITIALIZE does not affect index data items and FILLER data items.

5. INITIALIZE does not affect items subordinate to fld-name that contain a REDEFINES clause. Nor does it affect data items subordinate to those items. However, fld-name itself can have a REDEFINES clause or be subordinate to a data item that does.

6. When there is a REPLACING phrase, val is the sending field for each of the implicit MOVE statements.

7. When there is no REPLACING phrase, the sending field for the implicit MOVE statements is as follows:
   - SPACES, if the data item category is alphabetic, alphanumeric, or alphanumeric edited
   - ZEROS, if the data item category is numeric or numeric edited

8. INITIALIZE operates on each fld-name in the order it appears in the statement. When fld-name is a group item, INITIALIZE operates on its eligible subordinate elementary items in the order they are defined in the group.

9. If fld-name occupies the same storage area as val, the execution result of this statement is undefined. (See the section on Overlapping Operands and Incompatible Data.)

**Additional References**

- MOVE statement
- Section 6.6.7: Overlapping Operands and Incompatible Data

**Examples**

In the examples' results, a hyphen (-) means that the value of the data item is unchanged; s represents the character space. The examples assume this data description:

```plaintext
01  ITEMA.
   03  ITEMB      PIC X(4).
   03  ITMCM.
      05  ITEMD    PIC 9(5).
      05  ITME     PIC $$$9.99.
      05  ITMF     PIC XX/XX.
```
INITIATE

Function

The INITIATE statement causes the Report Writer Control System (RWCS) to begin processing a report.

[report-name]

names a report defined by a Report Description entry (RD) in the Report Section of the Data Division.

General Rules

1. The INITIATE statement does not automatically open a report file. The program must execute either an OPEN OUTPUT or an OPEN EXTEND statement before it can execute an INITIATE statement.

2. Upon execution of the INITIATE statement, the RWCS sets all sum counters, LINE-COUNTER, and PAGE-COUNTER to zero.

3. If the INITIATE statement has more than one report-name, the statement executes as if there were a separate INITIATE statement for each report-name.

4. A program must execute a TERMINATE statement before it can execute another INITIATE statement for the same report-name.

Additional Reference

USE statement
INSPECT

Function

The INSPECT statement counts or replaces occurrences of single characters or groups of characters in a data item.

[src-string]

is the identifier of a group item or an elementary data item with DISPLAY usage. INSPECT operates on the contents of this data item.

[tally-ctr]

is the identifier of an elementary numeric data item.

[delim-val]

is the character-string that delimits the INSPECT operation. Its content restrictions are the same as those for compare-val.

[compare-val]

is the character-string INSPECT uses for comparison. It is a nonnumeric literal (or figurative constant other than ALL literal) or the identifier of an elementary alphabetic, alphanumeric, or numeric data item with DISPLAY usage.

[replace-char]

is the one-character item that replaces all characters. Its content restrictions are the same as those for compare-val.

[replace-val]

is the character-string that replaces occurrences of compare-val. Its content restrictions are the same as those for compare-val.

[compare-chars]

is the string that contains the individual characters that convert to those in convert-chars. It is the same kind of item as compare-val.

[convert-chars]

is the string that contains the individual characters to which the characters in compare-chars convert. It is the same kind of item as compare-val.

Syntax Rules

All Formats

1. If compare-val, delim-val, replace-char, or compare-chars is a figurative constant, it refers to an implicit one-character data item.

2. A compare-val of an ALL or LEADING phrase, and a CHARACTERS, FIRST, or CONVERTING phrase can have no more than one BEFORE and one AFTER phrase following it.

Format 2

3. The sizes of the data referred to by replace-val and compare-val must be equal. When replace-val is a figurative constant, its size equals that of the data referred to by compare-val.
When there is a CHARACTERS phrase, the size of the data referred to by *delim-val* must be one character.

### Format 3

5. A Format 3 INSPECT statement is equivalent to a Format 1 statement followed by a Format 2 statement. Therefore, Syntax Rules 3 and 4 apply to the REPLACING clause of Format 3.

### Format 4

6. The sizes of the data referred to by *convert-chars* and *compare-chars* must be equal. When *convert-chars* is a figurative constant, its size equals that of the data referred to by *compare-chars*.

7. The same character cannot appear more than once in the data referred to by *compare-chars*.

### General Rules

#### All Formats

1. Inspection includes: (a) comparison, (b) setting boundaries for the BEFORE and AFTER phrases, and (c) tallying or replacing. Inspection starts at the leftmost character position of the *src-string* data item. It proceeds to the rightmost character position, as described in General Rules 3 to 5.

2. If *src-string*, *compare-val*, *delim-val*, *replace-val*, *compare-chars*, or *convert-chars* refers to a data item, the INSPECT statement treats the contents of the item according to the category implied by its data description.
   a. For an alphabetic or alphanumeric item – INSPECT treats the data item as a character-string.
   b. For an alphanumeric edited, numeric edited, or unsigned numeric item – INSPECT treats the data item as though:
      • The data item were redefined as alphanumeric.
      • The INSPECT statement were written to refer to the redefined data item. (See General Rule 2a.)
   c. For a signed numeric item – INSPECT treats the data item as though it were moved to an unsigned numeric data item of the same length. It then applies General Rule 2b.

3. If the size of *src-string* is zero characters, inspection does not occur.

4. If the size of *compare-val* is zero characters, *compare-val* does not match in any *src-string* comparison.

5. If any identifier is subscripted or is a function-identifier, the subscript or function-identifier is evaluated only once as the first operation in the execution of the INSPECT statement.

6. During inspection of *src-string*, each matched occurrence of *compare-val* is:
   a. Tallied (Formats 1 and 3)
   b. Replaced by *replace-char* or *replace-val* (Formats 2 and 3)

7. The comparison operation determines which occurrences of *compare-val* are tallied or replaced:
   a. INSPECT processes the operands of the TALLYING and REPLACING phrases in the order they appear, from left to right. The first *compare-val* is compared to the same number of contiguous characters, starting with the leftmost character position in *src-string*, *compare-val* and the compared characters in *src-string* match if they are equal, character for character. Otherwise, they do not match.
   b. If the comparison of the first *compare-val* does not produce a match, the comparison repeats for each successive *compare-val* until either:
      • A match results
• There is no next compare-val

When there is no next compare-val, INSPECT determines the leftmost character position in src-string for the next comparison. This position is to the immediate right of the leftmost character position for the preceding comparison. The comparison cycle starts again with the first compare-val.

c. For each match, tallying, replacing, or both occur, as described in General Rules 9 to 17. INSPECT determines the leftmost character position in src-string for the next comparison. This position is to the immediate right of the rightmost character position that matched in the preceding comparison. The comparison cycle starts again with the first compare-val.

d. Inspection ends when the rightmost character position of src-string has either:

  • Participated in a match
  • Served as the leftmost character position

e. When the CHARACTERS phrase is present, INSPECT does not perform any comparison on the contents of src-string. The cycle described in General Rules 6a to 6d operates as if:

  • Inspection compares a one-character data item to each character in src-string
  • A match occurs for each comparison

8. The BEFORE phrase determines the character position in src-string that will be the final leftmost position in the comparison operation.

  a. Comparison occurs on src-string only:

    • From its leftmost character position
    • To, but not including, the first occurrence of delim-val

  b. The position of the first occurrence of delim-val in src-string is determined before the first comparison operation.

  c. If delim-val does not occur in src-string, the comparison operation proceeds as if there were no BEFORE phrase.

9. The AFTER phrase determines the character position in src-string that will be the first leftmost position in the comparison operation.

  a. Comparison occurs on src-string only:

    • From the character position to the immediate right of the rightmost character position of delim-val's first occurrence
    • To the rightmost position of src-string

  b. The position of the first occurrence of delim-val in src-string is determined before the first comparison operation.

  c. If delim-val is not in src-string, no match occurs, and inspection causes no tallying or replacement.

Format 1

10. Executing the INSPECT statement does not initialize the value of tally-ctr.

11. If the ALL phrase is present, the value of tally-ctr is incremented by one for each occurrence of compare-val in src-string.
12. If the LEADING phrase is present, the value of tally-ctr is incremented by one for each contiguous occurrence of compare-val in src-string. The leftmost occurrence of compare-val must be at the position where comparison begins in the first comparison cycle. Otherwise, no tallying occurs.

13. If the CHARACTERS phrase is present, the value of tally-ctr is incremented by one for each character matched in src-string (see General Rule 6e).

**Format 2**

14. The adjectives ALL, LEADING, and FIRST apply to succeeding BY phrases until the next adjective appears.

15. If the CHARACTERS phrase is present, each character matched in src-string is replaced by replace-char (see General Rule 6e).

16. When ALL is present, each occurrence of compare-val in src-string is replaced by replace-val.

17. When LEADING is present, each contiguous occurrence of compare-val in src-string is replaced by replace-val. The leftmost occurrence of compare-val must be at the position where comparison begins in the first comparison cycle. Otherwise, no replacement occurs.

18. When FIRST is present, the leftmost occurrence of compare-val in src-string is replaced by replace-val.

**Format 3**

19. A Format 3 INSPECT statement executes as if there were two successive INSPECT statements with the same src-string. Execution proceeds as if:
   - The first statement were a Format 1 statement with TALLING phrases identical to those in the Format 3 statement
   - The second statement were a Format 2 statement with REPLACING phrases identical to those in the Format 3 statement

   The General Rules for Formats 1 and 2 apply to the corresponding phrases in the Format 3 statement.

**Format 4**

20. A Format 4 statement executes as if:
   - It were a Format 2 INSPECT statement with a series of ALL phrases, one for each character of compare-chars
   - compare-val in each ALL phrase referred to a single character of compare-chars
   - replace-val in each ALL phrase referred to a single character of replace-chars

   The individual characters of compare-chars and replace-chars correspond by ordinal position in the data items.

**Additional Reference**

- MOVE

**Examples**

In the following examples, the initial values of COUNT1 and COUNT2 are zero.

1. TALLING phrase with BEFORE option:

   INSPECT ITEMA TALLYING COUNT1 FOR LEADING "L" BEFORE "A",
   COUNT2 FOR LEADING "A" BEFORE "L".
ITEMA | COUNT1 | COUNT2
LARGE | 1 | 0
ANALYST | 0 | 1

2. TALLYING phrase and REPLACING LEADING phrase with AFTER option:

```
INSPECT ITEMA TALLYING COUNT1 FOR ALL "L" "R"
REPLACING LEADING "A" BY "E" AFTER INITIAL "L".
```

ITEMA | COUNT1 | ITEMA
CALLAR | 3 | CALLAR
SALAMI | 1 | SALEMI
LATTER | 2 | LETTER

3. REPLACING ALL phrase with BEFORE option:

```
INSPECT ITEMA REPLACING ALL "A" BY "G" BEFORE "X".
```

ITEMA | ITEMA
ARXAX | GRXAX
HANDAX | HGNDGX
HANDAA | HGNDGG

4. TALLYING and REPLACING ALL phrases:

```
INSPECT ITEMA TALLYING COUNT1 FOR CHARACTERS AFTER "J"
REPLACING ALL "A" BY "B".
```

ITEMA | COUNT1 | ITEMA
ADJECTIVE | 6 | BDJECTIVE
JACK | 3 | JBCK
JUJMBAB | 5 | JUJMBBB

5. REPLACING ALL phrase:

```
INSPECT ITEMA REPLACING ALL "X" BY "Y", "B" BY "Z",
"W" BY "Q" AFTER "R".
```

ITEMA | ITEMA
RXXBQWY | RYYZQQY
YZACDWBR | YZACDWWZR
RAWRXEB | RAQRYEZ

1. REPLACING CHARACTERS phrase:

```
INSPECT ITEMA REPLACING CHARACTERS BY "B" BEFORE "A".
```

ITEMA | ITEMA
12RXZABCD | BBBBBBABCBD
12RXZBBCD | BBBBBBBBBB

2. REPLACING ALL phrase:
INSPECT ITEMA REPLACING ALL "A" BY "X" ALL "R" BY "X"
AFTER "XXL".

<table>
<thead>
<tr>
<th>ITEMA</th>
<th>ITEMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>AALRRRA</td>
<td>XXLRRRX</td>
</tr>
<tr>
<td>AXXLRRR</td>
<td>XXXLXXX</td>
</tr>
</tbody>
</table>

3. CONVERTING phrase:

INSPECT ITEMA CONVERTING "SIR" TO "DTA"
AFTER QUOTE BEFORE "@".

<table>
<thead>
<tr>
<th>ITEMA</th>
<th>ITEMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIRMS &quot;SRXIL@STAR</td>
<td>TIRMS &quot;DAXTL@STAR</td>
</tr>
</tbody>
</table>

**MERGE**

**Function**

The MERGE statement takes two or more identically sequenced files and combines them according to the key values you specify. During the process, it makes records available, in merged order, to routines in OUTPUT PROCEDURE or to an output file.

[mergefile]

is a file-name described in a sort-merge file description (SD) entry in the Data Division.

[mergekey]

is the data-name of a data item in a record associated with *mergefile*.

[alpha]

is an alphabet-name defined in the SPECIAL-NAMES paragraph of the Environment Division.

[infile]

is the file-name of an input file. It must be described in a file description (FD) entry in the Data Division.

[first-proc]

is the section-name or paragraph-name of the output procedure's first (or only) section or paragraph.

[end-proc]

is the section-name or paragraph-name of the output procedure's last section or paragraph.

[outfile]

is the file-name of an output file. It must be described in a file description (FD) entry in the Data Division.

**Syntax Rules**

1. MERGE statements can appear anywhere in the Procedure Division except in:
• DECLARATIVES

• Sections of a SORT or MERGE statement's INPUT or OUTPUT PROCEDURE

2. If *mergefile* contains variable length records, *infile* records must not be smaller than the smallest record in *mergefile* nor larger than the largest.

3. If *mergefile* contains fixed-length records, *infile* records must not be larger than the largest record described for *mergefile*.

4. If *outfile* contains variable length records, *mergefile* records must not be smaller than the smallest record in *outfile* nor larger than the largest.

5. If *outfile* contains fixed-length records, *mergefile* records must not be larger than the largest record described for *outfile*.

6. Each *mergekey* must be described in records associated with *mergefile*.

7. *mergekey* can be qualified.

8. *mergekey* cannot be a group that contains variable occurrence data items.

9. The description of *mergekey* cannot contain an OCCURS clause or be subordinate to one that does.

10. *mergefile* can have more than one record description. However, *mergekey* need not be described in more than one of the record descriptions. The character positions referenced by *mergekey* are used as the key for all the file's records.

11. The words THRU and THROUGH are equivalent.

12. If *outfile* is an indexed file, the first *mergekey* must be in the ASCENDING phrase. It must specify the same character positions in its record as the prime record key for *outfile*.

**General Rules**

1. The MERGE statement merges all records in the *infile* files.

2. If *mergefile* contains fixed-length records, any shorter *infile* records are space-filled on the right after the last character. Space-filling occurs before the *infile* record is released to *mergefile*.

3. The leftmost *mergekey* is the major key, and the next *mergekey* is the next most significant key. The significance of *mergekey* data items is not affected by how they are divided into KEY phrases. Only left-to-right order determines significance.

4. The ASCENDING phrase causes the merged sequence to be from the lowest *mergekey* value to the highest.

5. The DESCENDING phrase causes the merged sequence to be from the highest *mergekey* value to the lowest.

6. Merge sequence follows the rules for relation condition comparisons.

7. When the contents of all key data items of one record equal the contents of the corresponding key data items in another record, the order of return from the merge:
   • Follows the order of the associated input files in the MERGE statement
   • Causes all records with equal key values from one input file to be returned before any are returned from another

8. The MERGE statement determines the comparison collating sequence for nonnumeric *mergekey* items when it begins execution. If there is a COLLATING SEQUENCE phrase in the MERGE statement, MERGE uses
that sequence. Otherwise, it uses the collating sequence that was established for the program as a whole in the
PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph. If you do not specify
the collating sequence in either the MERGE statement or the OBJECT-COMPUTER paragraph, the program
uses the NATIVE collating sequence.

9. The results of the merge are undefined unless the records in the infile files are ordered as described in the
MERGE statement's ASCENDING or DESCENDING KEY clause.

10. The MERGE statement transfers all records in infile to mergefile. When the MERGE statement executes, infile
must not be open.

11. For each infile, the MERGE statement:
   • Begins file processing as if the program had executed an OPEN statement with the INPUT phrase.
   • Obtains the logical records and releases them to the merge operation. MERGE obtains each record as if the
     program had executed a READ statement with the NEXT and AT END phrases.
   • Terminates file processing as if the program had executed a CLOSE statement with no optional phrases.

   These implicit OPEN, READ, and CLOSE operations cause associated USE procedures to execute if an
exception condition occurs.

12. OUTPUT PROCEDURE consists of one or more sections that are as follows:
   • Contiguous in the source program
   • Not a part of any other procedure

13. When the MERGE statement enters the OUTPUT PROCEDURE range, it is ready to select the next record in
merged order. Statements in the OUTPUT PROCEDURE range must execute at least one RETURN statement
to make records available for processing.

14. The OUTPUT PROCEDURE can consist of any procedure needed to select, modify, or copy the next record
made available by the RETURN statement in merged order from the file referenced by mergefile.

15. The range of the OUTPUT PROCEDURE additionally includes all statements executed as a result of a CALL,
EXIT, GO TO, or PERFORM statement. The range of the OUTPUT PROCEDURE also includes all statements
in the Declaratives Section that can be executed if control is transferred from statements in the range of the
OUTPUT PROCEDURE.

16. The range of the OUTPUT PROCEDURE must not contain MERGE, SORT, or RELEASE statements.

17. If the MERGE statement is in a fixed segment, the OUTPUT PROCEDURE range must be either:
   • Completely in fixed segments
   • Completely contained in one independent segment

18. If the MERGE statement is in an independent segment, the OUTPUT PROCEDURE range must be either:
   • Completely in fixed segments
   • Completely contained in the same independent segment as the MERGE statement itself

19. If OUTPUT PROCEDURE is used, control passes to its sections during execution of the MERGE statement.
   When control passes to the last statement in the OUTPUT PROCEDURE range, the MERGE statement ends.
   Control then transfers to the next executable statement after the MERGE statement.

20. During execution of statements in the OUTPUT PROCEDURE range – or any USE AFTER EXCEPTION
procedure implicitly invoked during the MERGE statement – no statement outside the range can manipulate
the files or record areas associated with infile or outfile.
21. If there is a GIVING phrase, the MERGE statement writes all merged records to each *outfile*. This transfer is an implied MERGE statement OUTPUT PROCEDURE. Therefore, when the MERGE statement executes, *outfile* must not be open.

22. The MERGE statement begins *outfile* processing as if the program had executed an OPEN statement with the OUTPUT phrase.

23. The MERGE statement gets the merged logical records and writes them to each *outfile*. MERGE writes each record as if the program had executed a WRITE statement with no optional phrases.

   For relative files, the value of the relative key data item is 1 for the first returned record, 2 for the second, and so on. When the MERGE statement ends, the value of the relative key data item indicates the number of *outfile* records.

24. The MERGE statement terminates *outfile* processing as if the program had executed a CLOSE statement with no optional phrases.

25. These implicit OPEN, WRITE, and CLOSE operations cause associated USE procedures to execute if an exception condition occurs. If the MERGE statement tries to write beyond the boundaries of *outfile*, the applicable USE AFTER EXCEPTION procedure executes. If control returns from the USE procedure, or if there is no USE procedure, *outfile* processing terminates as if the program had executed a CLOSE statement with no optional phrases.

26. If *outfile* contains fixed-length records, any shorter mergefile records are space-filled on the right after the last character. Space-filling occurs before the mergefile record is released to *outfile*.

**Additional References**

- OBJECT-COMPUTER paragraph in Chapter 4, *Environment Division*
- SPECIAL-NAMES paragraph in Chapter 4, *Environment Division*
- I-O-CONTROL paragraph in Chapter 4, *Environment Division*
- USE statement

**MOVE**

**Function**

The MOVE statement transfers data to one or more data areas. The editing rules control data transfer.

[src-item]

is an identifier that represents the sending area.

[lit]

is a literal that represents the sending area.

[dest-item]

is an identifier that represents the receiving area.

**Syntax Rules**

1. CORR is an abbreviation for CORRESPONDING.
2. In the CORRESPONDING phrase, both src-item and dest-item must be group items, and there can be only one dest-item.

3. If any dest-item is numeric or numeric edited, lit cannot be any of the following:
   - HIGH-VALUE
   - HIGH-VALUES
   - LOW-VALUE
   - LOW-VALUES
   - SPACE
   - SPACES
   - QUOTE
   - QUOTES

4. If lit is the figurative constant ALL literal and the usage of dest-item is COMP-1 or COMP-2, the MOVE statement uses only one occurrence of literal.

5. No operand can be an index data item.

**General Rules**

1. In Format 2, when the CORRESPONDING phrase is present, selected items in src-item are moved to selected items in dest-item. The rules for the CORRESPONDING option control these moves. The results are the same as if the MOVE statement were replaced by separate MOVE statements for each pair of corresponding items in src-item and dest-item.

2. In Format 1, the MOVE statement moves the sending area to the first dest-item, then to each additional dest-item, in the same order in which they appear in the statement.

3. If src-item is reference-modified, subscripted, or indexed, or is a function-identifier, the reference modifier, subscript, index, or function-identifier is evaluated once, immediately before the move to the first dest-item.

4. Subscript or index evaluation for a dest-item occurs immediately before the move to that item.

5. The length of src-item is evaluated once, immediately before the move to the first dest-item.

6. The length of each dest-item is evaluated immediately before the move to that item.

7. The result of the first of the following MOVE statements is equivalent to the three that follow. The word temp represents an intermediate result item supplied by the compiler.

   MOVE ITEMA (ITEMB) TO ITEMB, ITEMC (ITEMB).
   MOVE ITEMA (ITEMB) TO temp.
   MOVE temp TO ITEMB.
   MOVE temp TO ITEMC (ITEMB).

**Elementary Moves**

8. A move is elementary when dest-item is an elementary item, and the sending area is either an elementary data item or a literal.
   a. An elementary item belongs to one of these categories, depending on its PICTURE clause:
      - Numeric
• Alphabetic
• Alphanumeric
• Numeric edited
• Alphanumeric edited

b. Numeric literals are numeric. Nonnumeric literals are alphanumeric.

c. The figurative constant ZERO is numeric when moved to a numeric or numeric edited item. Otherwise, it is alphanumeric.

d. The figurative constant SPACE is alphabetic.

e. All other figurative constants are alphanumeric.

9. These rules apply to elementary moves between categories:

a. The figurative constant SPACE, or an alphanumeric edited or alphabetic data item, cannot be moved to a numeric or numeric edited data item.

b. A numeric literal, the figurative constant ZERO, or a numeric or numeric edited data item, cannot be moved to an alphabetic data item.

c. A noninteger numeric literal or data item cannot be moved to an alphanumeric or alphanumeric edited data item.

d. All other elementary moves are valid.

**Editing, De-Editing, and Data Conversion During Elementary Moves**

10. Editing, de-editing, or other required internal data conversions occur during elementary moves. They are controlled by the description of *dest-item*.

11. When *dest-item* is alphanumeric or alphanumeric edited, alignment and space-filling occur according to the Standard Alignment Rules.

   If *lit* or *src-item* is signed numeric, the operational sign is not moved. If the operational sign occupies a separate character position:

   a. The sign character is not moved.

   b. The size of *lit* or *src-item* is considered to be one less than its actual size (in terms of Standard Data Format characters).

   If the sending operand is numeric and contains the PICTURE symbol (P), all digit positions specified with this symbol are considered to have the value zero and are counted in the size of the sending operand.

12. When *dest-item* is numeric or numeric edited, decimal point alignment and zero-filling occur according to the Standard Alignment Rules.

   a. When *dest-item* is a signed numeric item, the sign from *lit* or *src-item* is placed in it. If the sending item is unsigned, a positive sign is placed in *dest-item*.

   b. When *dest-item* is an unsigned numeric item, the absolute value of *lit* or *src-item* is moved.

   c. When *lit* or *src-item* is alphanumeric, the move occurs as if the sending item were described as an unsigned numeric integer.
d. When src-item is numeric edited, the compiler de-edits it before moving it to dest-item. Src-item can be signed.

13. When dest-item is alphabetic, justification and space-filling occur according to the Standard Alignment Rules.

**Nonelementary Moves**

14. A nonelementary move occurs as if it were an alphanumeric-to-alphanumeric elementary move. However, there is no internal data conversion. The move is not affected by individual elementary or group items in either src-item or dest-item, except as noted in the General Rules for the OCCURS clause.

**Summary**

Table 6.13, “Valid MOVE Statements” summarizes the valid types of MOVE statements. References after slash marks show the applicable General Rule. For example, moving a numeric edited item to an alphabetic item is invalid because of General Rule 9b.

**Table 6.13. Valid MOVE Statements**

<table>
<thead>
<tr>
<th>Category of Sending Data Item (lit or src-item)</th>
<th>Category of Receiving Data Item (dest-item)</th>
<th>Alphabetic</th>
<th>Alphanumeric Edited Alphanumeric</th>
<th>Numeric Integer</th>
<th>Numeric Noninteger</th>
<th>Numeric Edited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabetic</td>
<td></td>
<td>Yes/13</td>
<td>Yes/11</td>
<td>No/9a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alphanumeric</td>
<td></td>
<td>Yes/13</td>
<td>Yes/11</td>
<td>Yes/12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alphanumeric Edited</td>
<td></td>
<td>Yes/13</td>
<td>Yes/11</td>
<td>No/9a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Numeric Integer</td>
<td></td>
<td>No/9b</td>
<td>Yes/11</td>
<td>Yes/12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Numeric Noninteger</td>
<td></td>
<td>No/9b</td>
<td>No/9c</td>
<td>Yes/12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Numeric Edited</td>
<td></td>
<td>No/9b</td>
<td>Yes/11</td>
<td>Yes/12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Additional References**

- Section 5.2.2: COBOL Standard Alignment Rules
- OCCURS clause in Chapter 5, *Data Division*
- PICTURE clause in Chapter 5, *Data Division*
- SIGN clause in Chapter 5, *Data Division*
- Section 6.6.5: CORRESPONDING Phrase

**Examples**

The following examples show the result of executing the statement:

MOVE ITEMA TO ITEMB.

An s indicates a space character.

- Numeric edited receiving item:
  
  (The PICTURE of ITEMA is S9999V99.)

<table>
<thead>
<tr>
<th>ITEMA Value</th>
<th>ITEMB PICTURE</th>
<th>ITEMB Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0023.00</td>
<td>ZZZZ.99</td>
<td>ss23.00</td>
</tr>
</tbody>
</table>
### Procedure Division

**ITEMA Value** | **ITEMB PICTURE** | **ITEMB Contents**
---|---|---

#### b.
-0036.93 | +++++.99 | s-36.93

#### c.
+1234.56 | ZZ,ZZZ,99 | 1,234.56

#### d.
+1234.56 | ZZ,ZZZ,99- | 1,234.56s

#### e.
+1234.56 | ZZ,ZZZ,99+ | 1,234.56+

#### f.
-1234.56 | $$,$$$,$$$,99DB | s$$1,234.56DB

#### g.
-1234.56 | $$,$$$,99- | s$$234.56-

#### h.
+0001.25 | $$,$$$,99 | s$$1.25

#### i.
-0001.25 | $$,$$$,99 | s$$1.25

#### j.
+0000.00 | $$,$$$,99 | s$$0.00

#### k.
+0000.00 | $$,$$$,$$$ | ssssssss

- **Alphanumeric receiving item:**

  (The PICTURE of ITEMA is X(4).)

<table>
<thead>
<tr>
<th><strong>ITEMA Value</strong></th>
<th><strong>ITEMB Description</strong></th>
<th><strong>ITEMB Contents</strong></th>
</tr>
</thead>
</table>

#### a.
ABCD | PIC X(4) | ABCD

#### b.
ABCD | PIC X(6) | ABCDss

#### c.
ABCD | PIC X(6) JUST | ssABCD

#### d.
ABCs | PIC X(6) JUST | ssABCs

#### e.
ABCD | PIC XXX | ABC

#### f.
ABCD | PIC XX JUST | CD

- **Alphanumeric edited receiving item:**

  (The PICTURE of ITEMA is X(7).)

<table>
<thead>
<tr>
<th><strong>ITEMA Value</strong></th>
<th><strong>ITEMB Description</strong></th>
<th><strong>ITEMB Contents</strong></th>
</tr>
</thead>
</table>

#### a.
063080s | XX/99/XX | 06/30/80

#### b.
30JUN80 | 99BAAAB99 | 30sJUNs80

#### c.
6374823 | XXXBXXX/XX/X | 637s482/3s/s

#### d.
123456s | 0XB0X0X0XB0 | 01s02s03s04s

- **Numeric edited sending item:**

<table>
<thead>
<tr>
<th><strong>ITEMA PICTURE</strong></th>
<th><strong>ITEMA Value</strong></th>
<th><strong>ITEMB PICTURE</strong></th>
<th><strong>ITEMB Value</strong></th>
</tr>
</thead>
</table>

#### a.
Z,ZZZ,99- | 1,234.56- | 999,999- | 234,560-

#### b.
ZZZ,ZZZ,99- | ss1,234.56- | $$$,$$$,99- | s$$1,234.56-

#### c.
$$,$$$,$$$,99CR | s$$1,234.56CR | $$$,99- | s$$1,234.56-

#### d.
$$,$$$,$$$,99DB | s$$1,234.56DB | ZZZ,ZZZ,99CR | ss1,234.56CR

#### e.
++++++.99 | +1234.56 | ZZZZZ,99+ | s1234.56+

#### f.
++++++.99 | #s-1234.56 | ZZZZZZ,99- | ss1234.56-

#### g.
−−−−−−.99 | -1234.56 | ZZZZZZ,99DB | s1234.56DB

#### h.
−−−−−−.99 | ss1234.56 | $$,$$$,99 | s$$1,234.56

#### i.
$$,$$$,99- | $123.45- | /XBXXBXXBXX/ | /$1s23.4s5-/ 

#### j.
$$,$$$,99- | $123.45- | /99B99B99B99/ | /00s00s01s23/
MULTIPLY

MULTIPLY

Function

The MULTIPLY statement multiplies two numeric operands and stores the product in one or more data items.

[num]

is a numeric literal or the identifier of an elementary numeric item.

[rsult]

is the identifier of an elementary numeric item. However, in Format 2, rsult can be an elementary numeric edited item. It is the resultant identifier.

[stment]

is an imperative statement executed when an on size error condition has occurred.

[stment2]

is an imperative statement executed when no on size error condition has occurred.

General Rules

1. In Format 1, the value of num is multiplied by the value of the first rsult. The product replaces the current value of the first rsult. The process repeats for each successive occurrence of rsult.

2. In Format 2, the values of the two operands before the word GIVING are multiplied together. The product replaces the current value of each rsult.

Additional References

• Section 6.1.4: Scope of Statements
• Section 6.6.1: Arithmetic Operations
• Section 6.6.3: ROUNDED Phrase
• Section 6.6.4: ON SIZE ERROR Phrase
• Section 6.6.7: Overlapping Operands and Incompatible Data
• Section 6.6.2: Multiple Receiving Fields in Arithmetic Statements

Examples

Each of the examples assume these data descriptions and beginning values:

INITIAL VALUES

03  ITEMA  PIC S99 VALUE 4.             4
03  ITEMB  PIC S99 VALUE -35.            -35
03  ITEMC  PIC S99 VALUE 10.             10
03  ITEMD  PIC S99 VALUE 5.              5

1. Without GIVING phrase:
RESULTS

MULTIPLY 2 BY ITEMB. ITEMB = -70

2. SIZE ERROR phrase:

(When the SIZE ERROR condition occurs, the values of the affected resultant identifiers do not change.)

MULTIPLY 3 BY ITEMB
ON SIZE ERROR ITEMB = -35
MOVE 0 TO ITEMC. ITEMC = 0

3. NOT ON SIZE ERROR phrase:

MULTIPLY 2 BY ITEMB
ON SIZE ERROR ITEMB = -70
MOVE 0 TO ITEMC
NOT ON SIZE ERROR
MOVE 1 TO ITEMC. ITEMC = 1

4. END-MULTIPLY and MULTIPLY results with SIZE ERROR:

(The stment in the SIZE ERROR phrase executes if any operation causes a size error condition. The first MULTIPLY statement terminates with END-MULTIPLY. The second MULTIPLY executes whether or not the SIZE ERROR condition occurs.)

MULTIPLY 4 BY ITEMA ITEMB ITEMC
ON SIZE ERROR
MOVE 1 TO ITEMD
END-MULTIPLY
MULTIPLY 2 BY ITEMA ITEMB ITEMC
ON SIZE ERROR
ADD 1 TO ITEMD
END-MULTIPLY.

<table>
<thead>
<tr>
<th>After First MULTIPLY</th>
<th>After Second MULTIPLY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEMA = #16</td>
<td>ITEMA = #32</td>
</tr>
<tr>
<td>ITEMB = -35</td>
<td>ITEMB = -70</td>
</tr>
<tr>
<td>ITEMC = #40</td>
<td>ITEMC = #80</td>
</tr>
<tr>
<td>ITEMD = #5</td>
<td>ITEMD = ###6</td>
</tr>
</tbody>
</table>

If the initial value of ITEMB had been -20, a SIZE ERROR condition would not have occurred during the first MULTIPLY. However, the second MULTIPLY would have caused the condition:

<table>
<thead>
<tr>
<th>After First MULTIPLY</th>
<th>After Second MULTIPLY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEMA = 16</td>
<td>ITEMA =##32</td>
</tr>
<tr>
<td>ITEMB = -80</td>
<td>ITEMB =#.80</td>
</tr>
<tr>
<td>ITEMC = 40</td>
<td>ITEMC =##80</td>
</tr>
<tr>
<td>ITEMD = #5</td>
<td>ITEMD =###6</td>
</tr>
</tbody>
</table>

OPEN

OPEN
Function

The OPEN statement creates an access stream to the file, makes the file available to the program, begins the processing of a file, and specifies file sharing.

[file-name]

is the name of a file described in the Data Division. It cannot be the name of a sort or merge file.

Leading and trailing blanks are removed from file specifications on all platforms before an OPEN is attempted. Embedded blanks and tabs are removed on OpenVMS systems only.

Syntax Rules

Format 1—Sequential, Line Sequential (Alpha, I64), Relative, or Indexed Files

1. The NO REWIND phrase can be used only for files with sequential organization.

2. The I-O phrase can be used only for mass storage files.

3. The I-O phrase cannot be used with LINE SEQUENTIAL.

4. The EXTEND phrase can be used for sequential access mode files only.

5. The WITH LOCK phrase cannot be used with the ALLOWING phrase, because it is invalid to specify both X/Open standard (WITH [NO] LOCK or LOCK MODE) and VSI standard (LOCK-HOLDING, ALLOWING, or REGARDLESS) file sharing for the same file connector.

Format 2—Report Writer Files

6. file-name must be in a file description entry containing a REPORT clause.

General Rules

All Files

1. Successful OPEN statement execution:
   • Creates an access stream to the file
   • Makes the file available to the program
   • Puts the file in an open mode
   • Associates the file with file-name through the file connector

2. An executable image can open a file-name more than once with the INPUT, OUTPUT, I-O, and EXTEND phrases. After the first OPEN statement, each later OPEN for the same file-name must follow the execution of a CLOSE statement for the file-name. However, the CLOSE statement must not have a REEL, UNIT, or LOCK phrase.

3. The OPEN statement does not get or release the first data record.

4. For an OPEN statement with the INPUT, I-O, or EXTEND phrases, file-name's file description entry must be equivalent to that used when the file was created.

5. The NO REWIND phrase applies only to sequential single-reel/unit files. If the concept of rewinding does not apply to the file's storage medium, then the open is successful and an I-O status is set.

6. If the file's storage medium allows rewinding, and:
• There is neither an EXTEND nor a NO REWIND phrase, then OPEN statement execution positions the file at its beginning.

• There is a NO REWIND phrase, then the OPEN statement does not reposition the file. The file must already be positioned at its beginning before the OPEN statement executes.

7. Successful execution of an OPEN statement sets the Current Volume Pointer to:

• The first or only reel/unit for an available input or input-output file

• The reel/unit containing the last logical record for an extend file

• The new reel/unit for an unavailable output, input-output, or extend file

8. If more than one file-name is in the OPEN statement, execution is the same as if there were a separate OPEN statement for each file-name.

9. A file's maximum record size is established when the file is created and must not subsequently be changed.

**Format 1—Sequential, Line Sequential (Alpha, I64), Relative, or Indexed Files**

10. A file is available if it is both:

• Physically present

• Recognized by the I-O system

Table 6.14, “Opening Available and Unavailable Sequential, Line Sequential (Alpha, I64), Relative, and Indexed Files” shows the result of opening available and unavailable sequential, relative, and indexed files.

**Table 6.14. Opening Available and Unavailable Sequential, Line Sequential (Alpha, I64), Relative, and Indexed Files**

<table>
<thead>
<tr>
<th>Open Mode</th>
<th>File Is Available</th>
<th>File Is Unavailable</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>Normal open</td>
<td>Error</td>
</tr>
<tr>
<td>INPUT (Optional File)</td>
<td>Normal open</td>
<td>Normal open</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The first read causes the at end condition or invalid key condition</td>
</tr>
<tr>
<td>I-O</td>
<td>Normal open</td>
<td>Error</td>
</tr>
<tr>
<td>I-O (Optional File)</td>
<td>Normal open</td>
<td>The OPEN creates the file</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>Creates a new version of the file</td>
<td>The OPEN creates the file</td>
</tr>
<tr>
<td></td>
<td>See General Rule 24</td>
<td></td>
</tr>
<tr>
<td>EXTEND</td>
<td>Normal open</td>
<td>Error</td>
</tr>
<tr>
<td>EXTEND (Optional File)</td>
<td>Normal open</td>
<td>The OPEN creates the file</td>
</tr>
</tbody>
</table>
11. Successful OPEN statement execution makes the file's record area available to the program. If the file connector is an external file connector, the file has only one record area for the executable image.

12. When a file is not in an open mode, no statement that references the file can execute either implicitly or explicitly, except for:
   - A MERGE statement
   - An OPEN statement
   - A SORT statement with the USING or GIVING phrase

13. An OPEN statement for a file must successfully execute before any allowable input-output statement executes for the file. Table 6.15, “Allowable Input-Output Statements for Sequential, Line Sequential (Alpha, I64), Relative, and Indexed Files” shows allowable input-output statements by file organization, access mode, and open mode for sequential, line sequential, relative, and indexed files.

### Table 6.15. Allowable Input-Output Statements for Sequential, Line Sequential (Alpha, I64), Relative, and Indexed Files

<table>
<thead>
<tr>
<th>File Organization</th>
<th>Access Mode</th>
<th>Statement</th>
<th>INPUT</th>
<th>OUTPUT</th>
<th>I-O</th>
<th>EXTEND</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQUENTIAL</td>
<td>SEQUENTIAL</td>
<td>READ</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REWRITE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRITE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNLOCK</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LINE (Alpha, I64)</td>
<td>SEQUENTIAL</td>
<td>READ</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REWRITE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRITE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNLOCK</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RELATIVE</td>
<td>SEQUENTIAL</td>
<td>DELETE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REWRITE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>START</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRITE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNLOCK</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RANDOM</td>
<td></td>
<td>DELETE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REWRITE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRITE</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>File Organization</td>
<td>Access Mode</td>
<td>Statement</td>
<td>Open Mode</td>
<td>INPUT</td>
<td>OUTPUT</td>
<td>I-O</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>--------------------</td>
<td>-----------</td>
<td>-------</td>
<td>--------</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UNLOCK</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>DYNAMIC</td>
<td>DELETE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ NEXT</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REWRITE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>START</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRITE</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNLOCK</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>INDEXED</td>
<td>SEQUENTIAL</td>
<td>DELETE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REWRITE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>START</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRITE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNLOCK</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RANDOM</td>
<td></td>
<td>DELETE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REWRITE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRITE</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNLOCK</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td></td>
<td>DELETE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ NEXT</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ PRIOR</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REWRITE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>START</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRITE</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNLOCK</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
14. If the file opened with the INPUT phrase is an optional file that is not present, the OPEN statement sets the File Position Indicator to indicate this condition.

15. An OPEN statement with the EXTEND phrase positions the file immediately after its last logical record. The definition of last logical record differs by file organization:
   - For sequential and line sequential files, it is the last record written in the file.
   - For relative files, it is the currently existing record with the highest relative record number.
   - For indexed files in ascending sort order, it is the currently existing record with the highest prime record key value.
   - For indexed files in descending sort order, it is the currently existing record with the lowest prime record key.
   - For Report Writer files, the last logical record is the last record written in the file.

16. Files for which the LINAGE clause has been specified must not be opened in the EXTEND mode.

17. The I-O phrase opens a mass storage file for both input and output operations.

18. When LOCK MODE IS AUTOMATIC or LOCK MODE IS MANUAL is specified and WITH LOCK is not specified, the file is shareable, and can be opened by more than one access stream (except for files opened in OUTPUT mode, which cannot be shared).

19. The NO OTHERS option or WITH LOCK option specifies exclusive file access by this access stream. The access stream created by the OPEN ALLOWING NO OTHERS or OPEN WITH LOCK statement has exclusive access to the file and, therefore, no other concurrent access stream can access (or open) the file.

20. The READERS option permits read-only access to the file for concurrent access streams.

21. The ALL, WRITERS, and UPDATERS phrases allow concurrent access streams access to the file.

22. If there is no ALLOWING phrase or WITH LOCK phrase, the default file-sharing behavior for files depends on the open mode and whether X/Open standard (Alpha or I64) or VSI standard file sharing is in effect.

   For files opened in input mode:
   - VSI standard—The default is ALLOWING READERS (see General Rule 21 for the exception).
   - X/Open standard (Alpha, I64)—The default is to make the file fully shareable.

   For files opened in modes other than input mode, the default is always to make the file exclusive. (Also see General Rule 24.)

The selection of X/Open (Alpha, I64) or VSI standard file-sharing default behavior is made as follows by the compiler:

   - On Alpha and I64, if X/Open standard syntax (LOCK MODE or WITH [NO] LOCK) has been specified for file-name prior to the OPEN statement, the compiler interprets the statement according to the X/Open standard.
   - If VSI standard syntax (LOCK-HOLDING, ALLOWING, or REGARDLESS) has been specified for file-name prior to the OPEN statement, the compiler interprets the statement according to the VSI standard.
   - If no file-sharing syntax (LOCK-HOLDING, ALLOWING, REGARDLESS, LOCK MODE, or WITH [NO] LOCK) has been specified for file-name prior to the OPEN statement, then the compiler uses the /STANDARD=NOENSION qualifier on OpenVMS Alpha and I64 (or the Tru64 UNIX equivalent -std
[no]xopen flag) to determine whether the OPEN INPUT statement is interpreted as X/Open or VSI standard: a setting of xopen selects the X/Open standard, whereas a setting of no xopen selects the VSI standard.

Any subsequent I-O locking syntax for the same file connector in your program must be consistent: X/Open standard locking (Alpha, I64) and VSI standard locking (implicit or explicit) cannot be mixed for the same file connector.

24. On Tru64 UNIX systems, files opened in OUTPUT mode adhere to the same file-sharing protocols as do files opened in the EXTEND and I-O modes. Access can be denied or granted depending on the file lock requested and the file lock held.

On OpenVMS systems, file sharing is limited for OUTPUT mode. A higher-numbered version is always created by default.

On Alpha and I64 systems, if X/Open standard file sharing is in effect, files opened in OUTPUT mode cannot be shared.

25. On Tru64 UNIX systems, when two file connectors in one process concurrently access the same physical file, a file-locked condition is not generated.

On OpenVMS systems, when two file connectors in one process concurrently access the same physical file, a file-locked condition might be generated.

26. For files specified with a MULTIPLE FILE TAPE clause:

- The NO REWIND phrase, if specified, is ignored.
- Any required rewinding or positioning of the reel (or device) is accomplished according to the relative position of the file as specified in the MULTIPLE FILE TAPE clause.

27. An OPEN OUTPUT statement for a file specified with a POSITION phrase of a MULTIPLE FILE TAPE clause is invalid unless the tape contains all the files at positions prior to the position specified.

28. An OPEN OUTPUT statement for a file specified with a POSITION phrase of a MULTIPLE FILE TAPE clause is invalid if the tape already contains a file at the position specified.

29. An OPEN INPUT statement for a file specified with a POSITION phrase of a MULTIPLE FILE TAPE clause is invalid unless the tape contains a file at that position, as well as all the files at the positions prior to the position specified.

30. A file specified in a MULTIPLE FILE TAPE clause cannot be opened in either I-O or EXTEND mode.

Format 2—Report Writer Files

31. A file is available if it is physically present and recognized by the I-O system.

Table 6.16, “Opening Available and Unavailable Report Writer Files” shows the results of opening available and unavailable Report Writer files.

Table 6.16. Opening Available and Unavailable Report Writer Files

<table>
<thead>
<tr>
<th>Open Mode</th>
<th>File Is Available</th>
<th>File Is Unavailable</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>Creates a new version of the file</td>
<td>The OPEN creates the file</td>
</tr>
<tr>
<td>EXTEND</td>
<td>Normal OPEN</td>
<td>The OPEN is unsuccessful</td>
</tr>
<tr>
<td>EXTEND (optional file)</td>
<td>Normal OPEN</td>
<td>The OPEN creates the file</td>
</tr>
</tbody>
</table>

32. Successful OPEN statement execution makes the file's record area available to the Report Writer Control System. If the file connector is an external file connector, the file has only one record area for the executable image.
33. When a file is not in an open mode, no statement that references the file can execute either implicitly or explicitly, except for the OPEN statement.


### Table 6.17. Allowable Statements for Report Writer Files

<table>
<thead>
<tr>
<th>Statement</th>
<th>OUTPUT</th>
<th>EXTEND</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIATE</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GENERATE</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SUPPRESS</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TERMINATE</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>All other I-O statements</td>
<td>No</td>
<td>No (for record I-O only)</td>
</tr>
</tbody>
</table>

#### Technical Notes

- OPEN statement execution can result in these FILE STATUS data item values:

<table>
<thead>
<tr>
<th>File Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Open is successful.</td>
</tr>
<tr>
<td>05</td>
<td>Optional file not present.</td>
</tr>
<tr>
<td>07</td>
<td>No rewind on non-reel device.</td>
</tr>
<tr>
<td>35</td>
<td>File not found.</td>
</tr>
<tr>
<td>37</td>
<td>An open in I-O mode is attempted for a nonmass storage file, or an open is attempted for nonmass storage file that was declared as a relative or indexed file.</td>
</tr>
<tr>
<td>38</td>
<td>An open is attempted on a file closed with lock.</td>
</tr>
<tr>
<td>39</td>
<td>A mismatch exists between the current program's description of an index key and the existent file's description of the key, or (for OpenVMS systems) there is a conflict of maximum record size or record type.</td>
</tr>
<tr>
<td>41</td>
<td>File is already open</td>
</tr>
<tr>
<td>91</td>
<td>Open is unsuccessful; file locked by another access stream.</td>
</tr>
<tr>
<td>95</td>
<td>No file space on device.</td>
</tr>
<tr>
<td>30</td>
<td>All other permanent errors.</td>
</tr>
</tbody>
</table>

You must use the I-O-CONTROL statement APPLY PREALLOCATION with a value greater than 0 (the default is 0) to enable the detection of "device full" (file status 95) with the OPEN statement.

- Attempts to specify both X/Open standard and VSI standard file-sharing syntax for the same file connector are invalid. When the compiler cannot detect such attempts because they occur in different compilation units, the run-time system detects and reports the violations (file status 30). This holds for explicit and implicit usage.

#### Additional References

- LOCK MODE clause in the FILE-CONTROL paragraph in Chapter 4, Environment Division
- I-O-CONTROL paragraph in Chapter 4, Environment Division
- CLOSE statement
PERFORM

Function

The PERFORM statement executes one or more intra-program procedures. It returns control to the end of the PERFORM statement when all procedures have completed execution.

[first-proc]

is a procedure-name that identifies a paragraph or section in the Procedure Division. The set of statements in first-proc is the first (or only) set of statements in the PERFORM range.

[end-proc]

is a procedure-name that identifies a paragraph or section in the Procedure Division. The set of statements in end-proc is the last set of statements in the PERFORM range.

[stment]

is an imperative statement.

[repeat-count]

is a numeric integer literal or the identifier of a numeric integer elementary item. It controls how many times the statement set (or sets) executes.

[cond]

can be any conditional expression.

[var]

is an index-name or the identifier of a numeric integer elementary data item. Its value is changed by increm each time all statements in the PERFORM range execute.

[init]

is a numeric literal, index-name, or the identifier of a numeric elementary data item. It specifies the value of var before any statement in the PERFORM range executes.

[increm]

is a nonzero numeric literal or the identifier of a numeric elementary data item. It systematically changes the value of var each time the program executes all statements in the PERFORM range.

Syntax Rules

All Formats

1. If there is no first-proc, the PERFORM statement must contain stment and the END-PERFORM phrase. If there is a first-proc, the statement cannot have the END-PERFORM phrase.
2. If either `first-proc` or `end-proc` is placed in the Declaratives part of the Procedure Division, both must also be placed in the same `DECLARATIVES` section.

3. The words `THRU` and `THROUGH` are equivalent and interchangeable.

### Formats 3 and 4

4. If there is no `TEST BEFORE` or `TEST AFTER` phrase, `TEST BEFORE` is the default.

#### Format 4

5. If there is no `first-proc`, there can be no `AFTER` phrase.

6. If `var` is an index-name:
   - `init` must be an integer data item or a positive integer literal
   - `increm` must be an integer data item or a nonzero integer literal

7. If `init` is an index-name:
   - `var` must be an integer data item
   - `increm` must be an integer data item or a positive integer literal

### General Rules

#### All Formats

1. When `first-proc` appears, the statement is an out-of-line `PERFORM` statement. Otherwise, it is an in-line `PERFORM` statement.

2. The statements in the range of `first-proc` to `end-proc` for an out-of-line `PERFORM` are the statement set. For an in-line `PERFORM`, the statement set is contained within the scope of the `PERFORM...END-PERFORM` syntax.

3. Unless restricted to in-line or out-of-line statements, all General Rules apply to both types of `PERFORM` statements. An in-line `PERFORM` statement operates according to the general rules for an out-of-line `PERFORM`, except for periods, which are not allowed within the body of the `PERFORM`. The statements in the in-line `PERFORM` execute in place of the statements in the range of `first-proc` to `end-proc`.

4. When the `PERFORM` statement executes, control transfers to the first statement of `first-proc`. However, for Format 2, 3, or 4 `PERFORM` statements, transfer of control depends on evaluation of the specified condition.

   Transfer of control occurs only once for each `PERFORM` statement executed. When transfer of control does occur, after the statement set executes, control implicitly transfers back to the to the end of the perform statement as follows:

   - If `first-proc` is a paragraph-name and there is no `end-proc`, the return is after the last statement of `first-proc`.
   - If `first-proc` is a section-name and there is no `end-proc`, the return is after the last statement of the last paragraph of `first-proc`.
   - If `end-proc` is a paragraph-name, the return is after the last statement of `end-proc`.
   - If `end-proc` is a section-name, the return is after the last statement of the last paragraph of `end-proc`.
   - If the statement is an in-line `PERFORM`, execution ends after the last statement of the statement set.

5. `first-proc` and `end-proc` need not be related except that `first-proc` is the beginning and `end-proc` is the last in a consecutive series of operations.
GO TO and PERFORM statements can occur between first-proc and end-proc. If there are two or more logical paths to the return point, end-proc can be a paragraph, consisting of the EXIT statement, to which all these paths must lead.

6. If a statement other than a PERFORM statement, transfers control to the statement set, at the end of the statement set, control transfers through the last statement of the set to the next executable statement following the set as if no PERFORM statement referenced the set.

7. The range of a PERFORM statement consists of all statements executed as a result of executing the PERFORM. It continues through execution of the implicit control transfer to the end of the PERFORM statement.

8. The range of the PERFORM statement additionally includes all statements executed as a result of a CALL, EXIT, GO TO, or PERFORM statement. The range the PERFORM statement also includes all statements in the Declaratives Section that can be executed if control is transferred from statements in the range of the PERFORM statement.

9. Statements executed as the result of a control transfer caused by an EXIT PROGRAM statement are not part of the range when:
   - The EXIT PROGRAM statement is specified in the same program as the PERFORM statement, and
   - The EXIT PROGRAM statement is within the range of the PERFORM statement.

10. A PERFORM statement in a fixed segment can have only one of the following in its range:
    - Sections and paragraphs completely contained in one or more nonindependent segments
    - Sections and paragraphs completely contained in one independent segment

    However, the PERFORM statement range also includes any Declarative procedures activated during its execution.

11. A PERFORM statement in an independent\(^1\) segment can have only one of the following in its range:
    - Sections and paragraphs completely contained in one or more nonindependent segments
    - Sections and paragraphs completely contained in the same independent segment as the PERFORM statement itself

    However, the PERFORM statement range also includes any Declarative procedures activated during its execution.

12. first-proc and end-proc cannot name sections or paragraphs in any other program in the executable image. Statements in other programs are in a PERFORM statement's range only if the range includes a CALL statement.

13. A PERFORM statement range can contain another PERFORM statement. In that case, the included PERFORM statement's sequence of procedures must be either totally included in, or excluded from, the logical sequence of the first PERFORM statement.

    For example:
    - An active PERFORM statement whose execution point is in the range of another active PERFORM statement must not allow control to pass to the exit of the other active PERFORM.
    - Two or more active PERFORM statements cannot have a common exit.

    Use the check compiler option with the perform keyword to verify at run time that there is no recursive activation of a PERFORM.

\(^1\)Segmentation is described in Section 6.7, “Segmentation”. VSI COBOL supports segmentation for compatibility with existing applications only. VSI recommends that you do not use segmentation in new applications.
Figure 6.2, “Valid and Invalid Nested PERFORM Statements” shows valid and invalid nested PERFORM statements.

14. Undocumented results might occur when end-proc precedes first-proc or when first-proc and end-proc are not in the same program segment.

**Figure 6.2. Valid and Invalid Nested PERFORM Statements**

**Format 1**

15. Format 1 is the basic PERFORM statement. The statement sets in the PERFORM range execute once. Control then passes to the end of the PERFORM statement.

**Format 2**

16. The statement sets execute the number of times specified by repeat-count. If the value of repeat-count is zero or negative when the PERFORM statement executes, control passes to the end of the PERFORM statement.

During PERFORM statement execution, changing the value of repeat-count does not change the number of times the statement sets execute.

**Format 3**

17. The statement sets execute until cond is true. Control then transfers to the end of the PERFORM statement.

18. If cond is true when the PERFORM statement executes:
   - If there is a TEST BEFORE phrase or one is implied, there is no transfer to first-proc; control passes to the end of the PERFORM statement.
   - If there is a TEST AFTER phrase, the PERFORM statement tests cond after the statement set executes.

**Format 4**

19. The Format 4 PERFORM statement systematically changes the value of var during its execution.

20. If var is an index-name, its value, when the PERFORM statement execution begins, must equal the occurrence number of an element in its table.

21. If init is an index-name, var must equal the occurrence number of an element in the table associated with init. As the value of the var index changes during PERFORM execution, it cannot contain a value outside the range of its table. However, when the PERFORM statement ends, the var index can contain a value outside the range of the table by one increment or decrement value.

22. increm must not be zero.

23. init must be positive when var is an index-name and init is an identifier.

24. If there is a TEST BEFORE phrase (or one is implied), and one var is varied (see Figure 6.3, “PERFORM ... VARYING with the TEST BEFORE Phrase and One Condition”):
   - var is set to the value of init when the PERFORM statement begins to execute.
   - If cond is false, the statement set executes once. The value of var changes by the increment or decrement value (increm), and cond is evaluated again. This cycle continues until cond is true. Control then transfers to the end of the PERFORM statement.
• If $cond$ is true when the PERFORM statement begins executing, control transfers to the end of the PERFORM statement.

**Figure 6.3. PERFORM … VARYING with the TEST BEFORE Phrase and One Condition**

25. If there is a TEST BEFORE phrase (or one is implied), and the PERFORM statement has two $vars$ (see Figure 6.4, “PERFORM … VARYING with the TEST BEFORE Phrase and Two Conditions”):

- The first and second $vars$ are set to the value of the first and second $init$ when the PERFORM statement begins to execute.

- If the first $cond$ is true, control transfers to the end of the PERFORM statement.

- If the second $cond$ is false, the statement set executes once. The second $var$ changes by the value of $increm$, and the second $cond$ is evaluated again. This cycle continues until the second $cond$ is true.

- When the second $cond$ is true, the value of the first $var$ changes by the value of the first $increm$, and the second $var$ is set to the value of the second $init$. The first $cond$ is reevaluated. The PERFORM statement ends if the first $cond$ is true. Otherwise, the cycle continues until $cond$ is true.

**Figure 6.4. PERFORM … VARYING with the TEST BEFORE Phrase and Two Conditions**

26. At the end of a PERFORM statement with the TEST BEFORE phrase:

- The value of the first $var$ exceeds the last-used value by one increment or decrement value. However, if $cond$ was true when the PERFORM statement began, $var$ contains the current value of $init$.

- The value of each other $var$ equals the current value of its associated $init$.

27. If there is a TEST AFTER phrase and one $var$ is varied (see Figure 6.5, “PERFORM … VARYING with the TEST AFTER Phrase and One Condition”):

- $var$ is set to the value of $init$ when the PERFORM statement starts to execute.

- The statement set executes once. Then, $cond$ is evaluated. If it is false, the value of $var$ changes by the increment or decrement value ($increm$), and the statement set executes again. This cycle continues until $cond$ is true. Control then transfers to the end of the PERFORM statement.

**Figure 6.5. PERFORM … VARYING with the TEST AFTER Phrase and One Condition**

28. If there is a TEST AFTER phrase, and two $vars$ are varied (see Figure 6.6, “PERFORM … VARYING with the TEST AFTER Phrase and Two Conditions”):

- The first and second $vars$ are set to the value of the first and second $init$ when the PERFORM statement starts to execute.

- The statement set executes. The second $cond$ is then evaluated. If it is false, the second $var$ changes by the value of $increm$, and the statement set executes again. This cycle continues until the second $cond$ is true.

- When the second $cond$ is true, the first $cond$ is evaluated. If it is false, the value of the first $var$ changes by the value of the first $increm$, the second $var$ is set to the value of the second $init$, and the statement set
executes again. The PERFORM statement ends if the first cond is true. Otherwise, the cycle continues until cond is true.

Figure 6.6. PERFORM … VARYING with the TEST AFTER Phrase and Two Conditions

29. At the end of a PERFORM statement with the TEST AFTER phrase, the value of each var is the same as at the end of the most recent statement set execution.

30. During execution of the sets of statements in the range, any change to var, increm, or init affects PERFORM statement operation.

31. When there is more than one var, var in each AFTER phrase goes through a complete cycle each time var in the preceding AFTER (or VARYING) phrase is varied.

Additional References

• Section 6.1.4: Scope of Statements
• Section 6.5: Conditional Expressions
• SET statement
• Online help, check compiler option

Examples

In the examples’ results, s represents a space. The examples assume these Data Division and Procedure Division entries:

DATA DIVISION.
   WORKING-STORAGE SECTION.
   01 ITEMA VALUE "ABCDEFGHIJ".
      03 CHARA OCCURS 10 TIMES PIC X.
   01 ITEMB VALUE SPACES.
      03 CHARB OCCURS 10 TIMES PIC X.
   01 ITEMC PIC 99 VALUE 1.
   01 ITemd PIC 99 VALUE 7.
   01 ITEME PIC 99 VALUE 4.
   01 ITEMF VALUE SPACES.
      03 ITemG OCCURS 4 TIMES.
         05 ITEMH OCCURS 5 TIMES.
         07 ITEMI PIC 99.

PROCEDURE DIVISION.
   .
   .
   .
   PROC-A.
      MOVE CHARA (ITEMC) TO CHARB (ITEMC).
   PROC-B.
      MOVE CHARA (ITEMC) TO CHARB (10).
   PROC-C.
      ADD 2 TO ITEMC.
   PROC-D.
MULTIPLY ITEMC BY ITEMD
GIVING ITEMI (ITEMC, ITEMD).

1. Performing one procedure (Format 1):
   PERFORM PROC-A.
   ITEMB = Asssssssss

2. Performing a range of procedures (Format 1):
   PERFORM PROC-A THRU PROC-B.
   ITEMB = AsssssssssA

3. Performing a range of procedures (Format 2):
   PERFORM PROC-A THRU PROC-C
   3 TIMES.
   ITEMB = AsCsEssssE
   ITEMC = 07

4. Performing a range of procedures (Format 4):
   PERFORM PROC-A THRU PROC-B
       VARYING ITEMC FROM 1 BY 1
       UNTIL ITEMC > 5.
   ITEMB = ABCDEssssE
   ITEMC = 06

5. Testing the UNTIL condition after execution (Format 4):
   PERFORM PROC-A THRU PROC-B
       TEST AFTER
       VARYING ITEMC FROM 1 BY 1
       UNTIL ITEMC > 5.
   ITEMB = ABCDEFssssF
   ITEMC = 06

6. Performing a range of procedures varying a data item by a negative amount (Format 4):
   PERFORM PROC-A THRU PROC-B
       VARYING ITEMC FROM ITEMD BY -1
       UNTIL ITEMC < ITEME.
   ITEMB = sssDEFGssD
   ITEMC = 03

7. In-line PERFORM (Format 4):
   PERFORM
       VARYING ITEMC FROM 1 BY 2
       UNTIL ITEMC > 7
       MOVE CHARA (ITEMC) TO CHARB (ITEMC)
       MOVE CHARA (ITEMC) TO CHARB (ITEMC + 3)
END-PERFORM.

ITEMB = AsCAECGEsG

8. Varying two data items (Format 4):

PERFORM PROC-D
  VARYING ITEMC FROM 1 BY 1 UNTIL ITEMC > 4
  AFTER ITEMD FROM 1 BY 1 UNTIL ITEMD > 5.

ITEMG (1) = 01s02s03s04s05s
ITEMG (2) = 02s04s06s08s10s
ITEMG (3) = 03s06s09s12s15s
ITEMG (4) = 04s08s12s16s20s

READ

READ

Function

For sequential access files, the READ statement makes the next logical record available. For random access files, READ makes a specified record available.

[file-name]

is the name of a file described in the Data Division. It cannot be a sort or merge file.

[dest-item]

is the identifier of a data item that receives the record accessed by the READ statement.

[stment]

is an imperative statement executed when the relevant condition (at end or invalid key) occurs.

[stment2]

is an imperative statement executed when the relevant condition (not at end or not invalid key) occurs.

[key-data]

is the data-name of a data item specified as a record key for file-name or the segmented-key name specified as a record key for file-name. It can be qualified.

Syntax Rules

1. Format 1 must be used for a sequential access mode file.

2. There must be a NEXT phrase for dynamic access mode files to retrieve records sequentially.

3. READ file-name PRIOR and READ file-name PREVIOUS are equivalent syntax.

4. Format 2 can be used for random or dynamic access mode files to retrieve records randomly.

5. The KEY phrase can be used only for indexed files.
6. To use the REGARDLESS or ALLOWING options the program must specify these entries:
   - APPLY LOCK-HOLDING clause of the I-O-CONTROL paragraph
   - ALLOWING option of the OPEN statement

7. There must be an INVALID KEY or AT END phrase when there is no applicable USE AFTER EXCEPTION procedure for the file.

8. The storage area associated with dest-item and the record area associated with file-name cannot be the same storage area.

9. On Alpha and I64 systems, the WITH [NO] LOCK phrase is X/Open standard syntax. It is invalid to specify both X/Open standard and VSI standard (LOCK-HOLDING, ALLOWING, OR REGARDLESS) file-sharing syntax for the same file connector. Hence, the WITH [NO] LOCK phrase cannot be used with the ALLOWING or REGARDLESS phrase.

General Rules

1. The file must be open in the INPUT or I-O mode when the READ statement executes.

2. For sequential access mode files, the NEXT phrase is optional. It has no effect on READ statement execution.

3. READ PRIOR can only be used with an INDEXED file whose organization is INDEXED and whose access mode is DYNAMIC. The file must be opened for INPUT or I-O.

4. Executing a Format 1 READ statement can cause the following to occur:
   - The record pointed to by the File Position Indicator becomes available in the file's record area.
   - For sequential and relative files, the File Position Indicator points to the file's next existing record.
   - For indexed files, the File Position Indicator points to the next existing record established by the file's Key of Reference.
   - If the file has no next record, the File Position Indicator indicates that no next logical record exists.

5. The READ statement updates the value of the FILE STATUS data item for the file.

6. A record is available before any statement executes after the READ.

7. More than one record description can describe a file's logical records. The records then share the same record area in storage. Sharing a record area is equivalent to implicit redefinition.
   READ statement execution does not change the contents of data items in the record area beyond the range of the current data record. The contents of those items are undefined.

8. A Format 1 READ statement can recognize the end of reel/unit during its execution. If it has not reached the logical end of the file, the READ statement performs a reel/unit swap. The Current Volume Pointer points to the file's next reel/unit.

9. During execution of a Format 2 READ statement, the File Position Indicator can indicate that an optional file is not present. The invalid key condition then exists, and READ statement execution is unsuccessful.

10. When a Format 1 READ statement executes, the File Position Indicator can indicate that:
   - There is no next logical record.
   - No valid next record has been established.
   - An optional file is not present.
   - The number of significant digits in the relative record number is larger than the relative key data items.
When the READ statement detects the no valid next record condition, the READ is unsuccessful.

When the READ statement detects one of the above conditions, not including the no valid next record condition:

- It updates the FILE STATUS data item for the file to indicate the at end condition.
- If the READ statement has an AT END phrase, control transfers to \textit{stmt}. No USE AFTER EXCEPTION procedure for the file executes.
- If there is no AT END phrase, a USE AFTER EXCEPTION procedure must be associated with the file. Control transfers to that procedure. Control returns from the USE AFTER EXCEPTION procedure to the next executable statement after the end of the READ statement.

When the at end condition occurs, execution of the READ statement is unsuccessful.

11. After the unsuccessful execution of a READ statement, the contents of the file's record area are undefined. If an optional file is not present, the File Position Indicator is unchanged; otherwise, it indicates that no valid next record has been established. For indexed files, the Key of Reference is undefined.

12. READ PRIOR retrieves a record from an Indexed file which logically precedes the one which was made current by the previous file access operation, if such a logically previous record exists.

13. For a relative or indexed file in dynamic access mode, a Format 1 READ statement with the NEXT phrase retrieves the file's next logical record. For an indexed file, when the Key of Reference has ascending sort order, the next logical record is the next record with a key value equal to or greater than the previous key value. When the Key of Reference has descending sort order, the next logical record is the next record with a key value equal to or less than the previous key value.

14. For a relative file, a Format 1 READ statement updates the contents of the file's RELATIVE KEY data item. The data item contains the relative record number of the available record.

15. For a relative file, a Format 2 READ statement sets the File Position Indicator to the record whose relative record number is in the file's RELATIVE KEY data item. Execution then continues as specified in General Rule 3. If the record is not in the file, the invalid key condition exists, and READ statement execution is unsuccessful.

16. When your program sequentially accesses an indexed file for records with duplicate record key values in the Key of Reference, those records are made available to your program in the same order in which they were created. The duplicate values can be created by execution of WRITE or REWRITE statements.

17. For an indexed file, a Format 2 READ statement with the KEY phrase establishes key-name as the Key of Reference for the retrieval. For a dynamic access mode file, the same Key of Reference applies to later retrievals by Format 1 READ statement executions for the file. The Key of Reference continues in effect until a new Key of Reference is established.

18. For an indexed file, a Format 2 READ statement without the KEY phrase establishes the prime record key as the Key of Reference for the retrieval. For a dynamic access mode file, the same Key of Reference applies to later retrievals by Format 1 READ statement executions for the file. The Key of Reference continues in effect until a new Key of Reference is established.

19. For an indexed file, a Format 2 READ statement compares the value in the Key of Reference with the value in the corresponding data item in the file's records. The comparison continues until the READ statement finds the first record with an equal value. The READ statement sets File Position Indicator to the record. Execution then continues as specified in General Rule 3. If the READ statement cannot identify a record with an equal value, the invalid key condition exists. READ statement execution is then unsuccessful.

20. The Format 2 READ verb can use the KEY IS syntax to establish the key field within the file record which is the Key of Reference. An immediately subsequent READ PRIOR will follow the order of the Key of Reference
to access the logically previous record in the file according to that Key of Reference. If the KEY IS syntax is not used, the Key of Reference is understood to be the file's primary key field.

21. When a successful READ PRIOR has occurred and the Key of Reference has ascending order, the record retrieved can have the same key value or a smaller key value than the preceding record for the Key of Reference. If the Key of Reference has descending order, the record retrieved can have the same key value or a higher key value for the Key of Reference. The retrieved record can have the same key value if duplicate values for the Key of Reference exist on the file.

22. When a READ PRIOR has been executed and a logically previous record does not exist, a File Status value of 10 indicating END-OF-FILE is returned. A READ PRIOR which is done immediately after Opening the file will produce the END-OF-FILE status.

23. If the number of character positions in the record being read is less than the minimum size specified by the record description entries for the file, the record area to the right of the last valid character read is undefined.

24. The REGARDLESS and ALLOWING options can be used only in a VSI standard manual record-locking environment. To create a manual record-locking environment, an access stream must specify the APPLY LOCK-HOLDING clause of the I-O-CONTROL paragraph.

25. On Tru64 UNIX and OpenVMS, the REGARDLESS option enables an access stream to read a record regardless of any record locks held by other concurrent access streams. READ REGARDLESS holds no lock on the record read.

This statement generates a soft record lock condition if the record is locked by another access stream. This condition results in a File Status value of 90 and invokes an applicable USE procedure, if any. Execution of the READ REGARDLESS statement is considered successful and program execution resumes at the next statement following the READ REGARDLESS statement.

However, on Tru64 UNIX systems, the soft lock condition (file status 90) is not recognized for indexed files. A READ REGARDLESS statement for a record locked by another process performs the requested read operation on the record and returns a file status of 00.

26. The ALLOWING UPDATERS and WITH NO LOCK options permit other concurrent access streams in a manual record-locking environment to simultaneously READ, DELETE, START, and REWRITE the current record. These options hold no locks on the current record.

27. The ALLOWING READERS option permits other concurrent access streams in a VSI standard, manual record-locking environment to simultaneously read the current record. This option holds a read-lock on each such record read. No access stream can update the current record until it is unlocked.

On Tru64 UNIX systems, for indexed files, the ALLOWING READERS phrase has some limitations, which are described in the VSI COBOL User Manual (see the section on indicating access allowed to other streams in the chapter on sharing files).

28. The ALLOWING NO OTHERS or WITH LOCK option locks the record read by the current access stream. No other concurrent access stream can access this record until it is unlocked. Only this access stream can update this record. This option applies to files opened in I-O mode. See general rule 29.

29. For files opened for input, a READ statement does not acquire a record lock, regardless of the locking syntax specified. This applies to X/Open standard and VSI standard locking.

30. If there is an applicable USE AFTER EXCEPTION procedure, it executes whenever an input condition occurs that would result in a nonzero value in the first character of a FILE STATUS data item. However, it does not
execute if: (a) the condition is invalid key, and there is an INVALID KEY phrase or (b) the condition is at end, and there is an AT END phrase.

31. If no exception condition exists, the record is made available in the record area. Control is transferred to the end of the READ statement; however, if \texttt{stmt2} is specified, \texttt{stmt2} executes before control is transferred to the end of the READ statement.

**Technical Note**

- READ statement execution can result in these FILE STATUS data item values:

<table>
<thead>
<tr>
<th>File Status</th>
<th>File Organization</th>
<th>Access Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>All</td>
<td>All</td>
<td>Read is successful; record is available; lock acquired as requested</td>
</tr>
<tr>
<td>02</td>
<td>Ind</td>
<td>All</td>
<td>Read is successful; duplicate key detected</td>
</tr>
<tr>
<td>04</td>
<td>All</td>
<td>All</td>
<td>Record read larger or smaller than record area</td>
</tr>
<tr>
<td>10</td>
<td>All</td>
<td>Seq</td>
<td>No next logical record (at end), optional file not present (at end), or no valid next record (at end)</td>
</tr>
<tr>
<td>14</td>
<td>Rel</td>
<td>Seq</td>
<td>Relative record number too large for relative key data item (at end)</td>
</tr>
<tr>
<td>23</td>
<td>Ind, Rel</td>
<td>Rand</td>
<td>Record not in file (invalid key) or optional file not present (invalid key)</td>
</tr>
<tr>
<td>46</td>
<td>All</td>
<td>Seq</td>
<td>No valid next record</td>
</tr>
<tr>
<td>47</td>
<td>All</td>
<td>All</td>
<td>File not open, or incompatible open mode</td>
</tr>
<tr>
<td>90</td>
<td>All</td>
<td>All</td>
<td>Record locked by another user; record is available in record area; no lock is acquired (soft lock for VSI standard locking only)</td>
</tr>
<tr>
<td>92</td>
<td>All</td>
<td>All</td>
<td>Record locked by another user; record is not available; no lock is acquired (hard lock)</td>
</tr>
<tr>
<td>30</td>
<td>All</td>
<td>All</td>
<td>All other permanent errors</td>
</tr>
</tbody>
</table>
• On Alpha and I64 systems, use START before initiating a sequence of either READ NEXT statements or READ PRIOR/READ PREVIOUS statements. You should use START, if you switch between READ NEXT and READ PRIOR/READ PREVIOUS or vice versa.

• On Alpha and I64 systems, the order of duplicate key values for a descending key is not necessarily the same as the order of duplicate key values for READ PRIOR/READ PREVIOUS used with an ascending key defined as the same file record location as the descending key.

**Additional References**

• LOCK MODE clause in the FILE-CONTROL paragraph in Chapter 4, *Environment Division*

• I-O-CONTROL paragraph in Chapter 4, *Environment Division*

• Section 6.1.4: Scope of Statements

• Section 6.6.8: I-O Status

• Section 6.6.10: INVALID KEY Phrase

• Section 6.6.9: AT END Phrase

• Section 6.6.12: INTO Phrase

• OPEN statement

• UNLOCK statement

• USE statement

**RECORD (OpenVMS Only)**

**Function**

The RECORD statement creates an Oracle CDD/Repository dependency relationship between an VSI COBOL program and the Oracle CDD/Repository entity referred to by the RECORD statement.

```
[path-name]
```

is a partial or complete Oracle CDD/Repository path name. It specifies a dictionary entity in CDO format.

```
[relation-type]
```

is a valid Oracle CDD/Repository protocol. It specifies the type of relationship to be created between the VSI COBOL program and the CDO dictionary entity specified in the path name. The default relationship type is CDD SCOMPILED_DEPENDS_ON.

**Syntax Rules**

1. A space must precede the word RECORD.

2. The RECORD statement must be terminated by the separator period.

**General Rules**

1. *path-name* refers to the Oracle CDD/Repository path for a dictionary entity. The entity must be in CDO format.
2. The RECORD statement creates an Oracle CDD/Repository relationship between the VSI COBOL program through a compiled module entity (see Technical Notes) and the dictionary entity specified in the path name. This relationship information is then stored in Oracle CDD/Repository.

3. The RECORD statement is ignored unless the /DEPENDENCY_DATA compiler option is specified.

4. If the RECORD statement is in a contained program, the relationship created is between the outermost containing program and the entity specified in the path name.

**Technical Notes**

1. The *path-name* can be a nonnumeric literal or COBOL word formed according to the rules for user-defined names. It represents a complete or partial Oracle CDD/Repository path name specifying an Oracle CDD/Repository entity. If *path-name* is not a literal, the compiler translates hyphens in the COBOL word to underscore characters.

   The resultant path name must conform to all rules for forming Oracle CDD/Repository path names.

2. The *relation-type* can be a nonnumeric literal or COBOL word formed according to the rules for user-defined names. It must be a valid Oracle CDD/Repository protocol type. For example:
   - CDDSCOMPILED_DEPENDS_ON is an example of a COBOL word that is a valid Oracle CDD/Repository protocol type.
   - CDDSCOMPILED_DERIVED_FROM is an example of a nonnumeric literal that is also a valid Oracle CDD/Repository protocol type.

3. The RECORD statement creates a relationship between an Oracle CDD/Repository compiled module dictionary entity and the dictionary entity specified in the path name. A compiled module entity is automatically created and stored in Oracle CDD/Repository when the /DEPENDENCY_DATA compiler option is specified. The name of the compiled module entity is the *program-name* from the PROGRAM-ID paragraph, with hyphens translated to underscores.

**Additional References**

- The description of Oracle CDD/Repository in the VSI COBOL User Manual
- Oracle CDD/Repository Documentation Set

**RELEASE**

**RELEASE**

**Function**

The RELEASE statement transfers records to the initial phase of a sort operation.

```
[rec]
```

is the name of a logical record in a sort-merge file description (SD) entry. It can be qualified.

```
[src-area]
```

is the identifier of the data item that contains the data.

**Syntax Rules**

1. A RELEASE statement can be used only in an input procedure in the same program as the SORT verb. The input procedure must be associated with a SORT statement for the sort or merge file that contains *rec*. 
2. If `src-area` is a function-identifier, it must reference an alphanumeric function. When `src-area` is not a function-identifier, `rec` and `src-area` must not reference the same storage area.

**General Rules**

1. See the FROM Phrase section for a list of rules.

2. The RELEASE statement transfers the contents of `rec` to the first phase of the sort.

3. After the RELEASE statement executes, the record is no longer available in `rec` unless the associated sort or merge file-name is in a SAME RECORD AREA clause. In that case, the record is available to the program as a record of the sort-merge file-name. It is also available as a record of all other file-names in the same SAME RECORD AREA clause.

**Additional References**

- I-O-CONTROL paragraph in Chapter 4, *Environment Division*
- Section 6.6.11: FROM Phrase phrase

**RETURN**

RETURN

**Function**

The RETURN statement obtains sorted records from a sort operation. It also returns merged records in a merge operation.

`[smrg-file]`

is the name of a file described in a sort-merge file description (SD) entry.

`[dest-area]`

is the identifier of the data item to which the returned `smrg-file` record is moved.

`[stment]`

is an imperative statement executed for an at end condition.

`[stment2]`

is an imperative statement executed for a not at end condition.

**Syntax Rules**

1. A RETURN statement can be used only in an output procedure in the same program as the SORT verb. The output procedure must be associated with a SORT or MERGE statement for `smrg-file`.

2. The storage area associated with `dest-area` and the record area associated with `smrg-file` cannot be the same storage area.

**General Rules**

1. See the INTO Phrase section for a list of rules.
2. When more than one record description describes the logical records for `smrg-file`, the records share the same storage area. The contents of storage positions beyond the range of the returned record are undefined when the RETURN statement ends.

3. Before the output procedure executes, the File Position Indicator is updated. It points to the record whose key values make it first in the file. If there are no records, the File Position Indicator indicates the at end condition.

4. The RETURN statement makes the next record (pointed to by the File Position Indicator) available in the record area for `smrg-file`.

5. The File Position Indicator is updated to point to the next record in `smrg-file`. The key values in the SORT or MERGE statement determine the next record.

6. If `smrg-file` has no next record, the File Position Indicator is updated to indicate the at end condition.

7. If the File Position Indicator indicates the at end condition when the RETURN statement executes, `stmt` executes and control is transferred to the end of the RETURN statement. If the NOT AT END phrase is specified, it is ignored. The contents of the `smrg-file` record areas are undefined.

8. If the File Position Indicator does not indicate an at end condition when the RETURN statement executes, after the record is made available and after executing any implicit move resulting from the presence of an INTO phrase, control is transferred to `stmt2`, if specified. Otherwise, control is transferred to the end of the RETURN statement.

9. When the at end condition occurs:
   - RETURN statement execution is unsuccessful.
   - The File Position Indicator is not changed.

10. See the Scope of Statements section for a description of scope terminators such as END-RETURN.

**Additional References**
- Section 6.1.4: Scope of Statements
- I-O-CONTROL paragraph in Chapter 4, *Environment Division*
- Section 6.6.9: AT END Phrase
- Section 6.6.12: INTO Phrase

**REWRITE**

**Function**

The REWRITE statement logically replaces a mass storage file record.

```
[rec-name]
```

is the name of a logical record in the Data Division File Section. It can be qualified.

```
[src-item]
```

is the identifier of the data item that contains the data.
Procedure Division

[statement]

is an imperative statement executed for an invalid key condition.

[statement2]

is an imperative statement executed for a not invalid key condition.

Syntax Rules

1. To use the ALLOWING option, the program must include these entries:
   - APPLY LOCK-HOLDING clause of the I-O-CONTROL paragraph
   - ALLOWING clause of the OPEN statement

2. The INVALID KEY and the NOT INVALID KEY phrases cannot be specified in a REWRITE statement that refers to a sequential file or to a relative file with sequential access mode.

3. For a relative file with random or dynamic access mode, or for an indexed file, the REWRITE statement must have an INVALID KEY phrase when there is no applicable USE AFTER EXCEPTION procedure for the file.

4. If src-item is a function-identifier, it must reference an alphanumeric function. When src-item is not a function-identifier, rec-name and src-item must not reference the same storage area.

5. The ALLOWING clause is VSI standard file-sharing syntax, and cannot be used for a file connector that has had X/Open standard file-sharing syntax (WITH [NO] LOCK or LOCK MODE) specified.

General Rules

All Files

1. The file associated with rec-name must be a mass storage file. It must be open in the I-O mode when the REWRITE statement executes.
   Because line sequential files (Alpha, I64) cannot be opened in I-O mode, REWRITE cannot be used with line sequential files.

2. For sequential access mode files, the last input-output statement executed for the file before the REWRITE statement must be a successfully executed READ statement. The REWRITE statement logically replaces the record accessed by the READ.

3. The READ must lock the record for the REWRITE statement to be successful.

4. The record in rec-name is no longer available after the REWRITE statement successfully executes unless the associated file-name is in a SAME RECORD AREA clause. In this case, the record is also available to record areas of other file-names in the SAME RECORD AREA clause.

5. The REWRITE statement does not affect the File Position Indicator.

6. The REWRITE statement updates the value of the FILE STATUS data item for the file.

7. The ALLOWING option can be used only in a VSI standard, manual record-locking environment. To create a manual record-locking environment, the program must OPEN file-name with an ALLOWING option and specify the APPLY LOCK-HOLDING phrase of the I-O-CONTROL paragraph.

8. The ALLOWING option locks the current record rewritten by the current access stream. No other concurrent access stream can access this record until it is unlocked.
   However, on Tru64 UNIX systems, for indexed files the REWRITE statement with the ALLOWING clause does not acquire a record lock.
9. See the FROM Phrase section for a list of rules for the FROM phrase.

**Sequential Files**

10. The record named by `rec-name` must be the same size as the record being replaced.

**Relative Files**

11. For a random or dynamic access mode file, the REWRITE statement logically replaces the record specified in the RELATIVE KEY data item for `rec-name`'s file. If the record is not in the file, the invalid key condition exists. The update does not occur, and the data in the record area is not affected.

**Indexed Files**

12. For a sequential access mode file, the prime record key specifies the record to be replaced. The values of the prime record keys in the record to be replaced and the last record read from (or positioned in) the file must be equal.

13. For a random access mode file, the prime record key specifies the record to replace. If the program specifies duplicates on the prime record key, then it can replace only the first occurrence of a key value using random access mode. Replacing subsequent records with the same prime key value is done by sequentially positioning to the desired record in sequential or dynamic access mode.

14. For indexed files in dynamic access mode, the presence of DUPLICATES on the prime record key determines the behavior. If DUPLICATES are allowed, Rule 11 applies. If DUPLICATES are not allowed, Rule 12 applies.

15. For a record with an alternate record key:

- When the REWRITE does not change the value of an alternate record key, the order of retrieval is unchanged when the key is the Key of Reference.

- When duplicate key values are allowed, and the value of an alternate record key changes, the later retrieval order of the record changes when the key is the Key of Reference. The record's logical position is last in the group of records with the same value in the alternate record key that changed.

16. Any of the following occurrences cause the invalid key condition:

- The access mode is sequential, and the values in the prime record keys of the record to replace and the last record read from (or positioned in) the file are not equal.

- The value in the prime record key does not equal that of any record in the file.

- The value in an alternate record key whose definition does not have a DUPLICATES clause equals that of a record already in the file.

The update does not occur, and the data in the record area is not affected.

17. If there is an applicable USE AFTER EXCEPTION procedure, it executes whenever an input or output condition occurs that would result in a nonzero value in the first character position of a FILE STATUS data item. However, the INVALID KEY phrase (if present) supersedes a USE AFTER EXCEPTION procedure when there is an invalid key condition. In this case, the USE AFTER EXCEPTION procedure does not execute.

See the rules for the INVALID KEY phrase, Section 6.6.10, “INVALID KEY Phrase”.

18. The number of character positions in the record to be updated must not be less than the lowest or greater than the highest number of character positions allowed by the RECORD VARYING clause. In either case, the REWRITE statement is unsuccessful and the following occurs:

- The updating operation does not take place.
Procedure Division

• The contents of the record area remain unaffected.
• The I-O status of the file is set to a value that indicates the cause of the condition.

Technical Notes

• REWRITE statement execution can result in these FILE STATUS data item values:

<table>
<thead>
<tr>
<th>File Status</th>
<th>File Organization</th>
<th>Access Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>All</td>
<td>All</td>
<td>Rewrite is successful.</td>
</tr>
<tr>
<td>02</td>
<td>Ind</td>
<td>All</td>
<td>Created duplicate primary or alternate key.</td>
</tr>
<tr>
<td>21</td>
<td>Ind</td>
<td>Seq</td>
<td>Primary key changed after read.</td>
</tr>
<tr>
<td>22</td>
<td>Ind</td>
<td>All</td>
<td>Duplicate primary or alternate key (invalid key).</td>
</tr>
<tr>
<td>23</td>
<td>Ind, Rel</td>
<td>Rand</td>
<td>Record not in file (invalid key).</td>
</tr>
<tr>
<td>43</td>
<td>All</td>
<td>Seq</td>
<td>No previous read or record not locked by prior READ or START.</td>
</tr>
<tr>
<td>44</td>
<td>All</td>
<td>All</td>
<td>Invalid record size.</td>
</tr>
<tr>
<td>49</td>
<td>All</td>
<td>All</td>
<td>File not open, or incompatible open mode.</td>
</tr>
<tr>
<td>92</td>
<td>Ind, Rel</td>
<td>All</td>
<td>Record locked by another user; record is not available.</td>
</tr>
<tr>
<td>30</td>
<td>All</td>
<td>All</td>
<td>All other permanent errors.</td>
</tr>
</tbody>
</table>

Additional References

• I-O-CONTROL paragraph in Chapter 4, Environment Division
• Section 6.1.4: Scope of Statements
• Section 6.6.8: I-O Status
• Section 6.6.10: INVALID KEY Phrase
• Section 6.6.11: FROM Phrase
• OPEN statement
• READ statement
• UNLOCK statement
• USE statement

SEARCH

SEARCH
Function

The SEARCH statement searches for a table element that satisfies a condition. It sets the value of the associated index to point to the table element.

[src-table]

is a table identifier.

[pointr]

is an index-name or the identifier of a data item described as USAGE INDEX, or an elementary numeric data item with no positions to the right of the assumed decimal point.

[cond]

is any conditional expression.

[stment]

is an imperative statement.

[elemnt]

is an indexed data-name. It refers to the table element against which the argument is compared.

[arg]

is the argument tested against each elemnt in the search. It is an identifier, a literal, or an arithmetic expression.

[cond-name]

is a condition-name.

Syntax Rules

Both Formats

1. src_table must not be subscripted, indexed, or reference modified. However, its description must contain an OCCURS clause with the INDEXED BY phrase.

2. If the END-SEARCH phrase is specified, the NEXT SENTENCE phrase must not be specified.

Format 2

1. src_table must contain the KEY IS phrase in its OCCURS clause.

2. Each cond-name must be defined as having only one value. The data-name associated with cond-name must be in the KEY IS phrase of the OCCURS clause for src-table.

3. Each elemnt:
   
   • Can be qualified
   
   • Must be indexed by the first index-name associated with src-table, in addition to other indexes or literals required for uniqueness
   
   • Must be in the KEY IS phrase of the OCCURS clause for src-table

4. Neither arg nor any identifier in its arithmetic expression can either:
• Be used in the KEY IS phrase of the OCCURS clause for src-table
• Be indexed by the first index-name associated with src-table

5. When element or the data-name associated with cond-name is in the KEY phrase of the OCCURS clause for src-table, each preceding data-name (or associated cond-name) in that phrase must also be referenced.

General Rules

Both Formats

1. After executing a stment that does not end with a GO TO statement, control passes to the end of the SEARCH statement.

2. src_table can be subordinate to a data item that contains an OCCURS clause. In that case, an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. The SEARCH statement modifies the setting of only the index-name for src-table (and pointr, if there is one).

A single SEARCH statement can search only one dimension of a table; therefore, you must execute SEARCH statements repeatedly to search through a multidimensional table. Before each execution, SET statements must execute to change the values of index-names that need adjustment.

Format 1

3. The Format 1 SEARCH statement searches a table serially, starting with the current index setting.

   a. The index-name associated with src-table can contain a value that indicates a higher occurrence number than is allowed for src-table. If the SEARCH statement execution starts when this condition exists, the search terminates immediately. If there is an AT END phrase, stment then executes. Otherwise, control passes to the end of the SEARCH statement.

   b. If the index-name associated with src-table indicates a valid src-table occurrence number, the SEARCH statement evaluates the conditions in the order they appear. It uses the index settings to determine the occurrence numbers of items to test.

If no condition is satisfied, the index-name for src-table is incremented to refer to the next occurrence. The condition evaluation process repeats using the new index-name settings. However, if the new value of the index-name for src-table indicates a table element outside its range, the search terminates as in General Rule 3a.

When a condition is satisfied:

• The search terminates immediately.

• The stment associated with the condition executes.

• The index-name remains set at the occurrence that satisfied the condition.

4. If there is no VARYING phrase, the index-name used for the search is the first index-name in the OCCURS clause for src-table. Other src-table index-names are unchanged.

5. If there is a VARYING phrase, pointr can be an index-name for src-table. (pointr is named in the INDEXED BY phrase of the OCCURS clause for src-table.) The search then uses that index-name. Otherwise, it uses the first index-name in the INDEXED BY phrase.

6. pointr also can be an index-name for another table. (pointr is named in the INDEXED BY phrase in the OCCURS clause for that table entry.) In this case, the search increments the occurrence number represented by pointr by the same amount, and at the same time, as it increments the occurrence number represented by the src-table index-name.
7. If `pointr` is an index data item rather than an index-name, the search increments it by the same amount, and at the same time, as it increments the `src-table` index-name. If `pointr` is not an index data item or an index-name, the search increments it by one when it increments the `src-table` index-name.

8. Example 3, "Serial search with two WHEN phrases," illustrates the operation of a Format 1 SEARCH statement with two WHEN phrases.

**Format 2**

9. A SEARCH ALL operation yields predictable results only when:

- The data in the table has the same order as described in the KEY IS phrase of the OCCURS clause for `src-table`.
- The contents of the keys in the WHEN phrase identify a unique table element.

10. If the WHEN phrase conditions are not satisfied for any index setting in the allowed range, control passes to the AT END phrase `stment`, if there is one, or to the end of the SEARCH statement. In either case, the setting of the `src-table` index-name is not predictable.

11. If all the WHEN phrase conditions are satisfied for an index setting in the allowed range, control passes to either `stment` or the next sentence, whichever is in the statement. The `src-table` index-name then indicates the occurrence number that satisfied the conditions.

12. The index-name used for the search is the first index-name in the OCCURS clause for `src-table`. Other `src-table` index-names are unchanged.

**Additional References**

- On Alpha and I64, SORT statement (Format 2, for table sorting), useful for SEARCH ALL (which presumes a sorted table)
- OCCURS statement
- Section 6.1.4: Scope of Statements
- Section 6.5: Conditional Expressions

**Examples**

The examples assume these Data Division entries:

```
01 CUSTOMER-REC.
  03 CUSTOMER-USPS-STATE PIC XX.
  03 CUSTOMER-REGION PIC X.
  03 CUSTOMER-NAME PIC X(15).
01 STATE-TAB.
  03 FILLER PIC X(153) VALUE "AK3AL5AR5AZ4CA4CO4CT1DC1DE1FL5GA5HI3 - "IA2ID3IL2IN2KS2KY5LA5MA1MD1ME1MI2MN2 - "MO5MS5MT3NC5ND3NE2NH1NJ1NM4NV4NY1OH2 - "OK4OR3PA1RI1SC5SD3TN5TX4UT4VA5VT1WA3 - "WI2WV5WY4".
01 STATE-TABLE REDEFINES STATE-TAB.
  03 STATES OCCURS 51 TIMES ASCENDING KEY IS STATE-USPS-CODE INDEXED BY STATE-INDEX.
  05 STATE-USPS-CODE PIC XX.
  05 STATE-REGION PIC X.
```
Procedure Division

01 STATE-NUM PIC 99.
01 STATE-ERROR PIC 9.
01 NAME-TABLE VALUE SPACES.
  03 NAME-ENTRY OCCURS 8 TIMES
    INDEXED BY NAME-INDEX.
    05 LAST-NAME PIC X(15).
    05 NAME-COUNT PIC 999.

1. Binary search:

   (The correctness of this statement's operation depends on the ascending order of key values in the source table.)

   INITIALIZE-SEARCH.
   MOVE "NH" TO CUSTOMER-USPS-STATE.

   SEARCH ALL STATES
   AT END
      MOVE 1 TO STATE-ERROR
      GO TO SEARCH-END
   WHEN STATE-USPS-CODE (STATE-INDEX) = CUSTOMER-USPS-STATE
      MOVE 0 TO STATE-ERROR
      MOVE STATE-REGION (STATE-INDEX) TO CUSTOMER-REGION.

   SEARCH-END.
   DISPLAY " ".
   DISPLAY "Customer State index number = " STATE-INDEX WITH CONVERSION
      " Region = " STATE-REGION (STATE-INDEX)
      " State Error Code = " STATE-ERROR.

   Following are the results of the binary search:
   Customer State index number = 31   Region = 1   State Error Code = 0

2. Serial search with WHEN phrase:

   INITIALIZE-SEARCH.
   MOVE "2" TO CUSTOMER-REGION.

   SEARCH-LOOP.
   SEARCH STATES
   AT END
      MOVE 1 TO STATE-ERROR
      GO TO SEARCH-END
   WHEN STATE-REGION (STATE-INDEX) = CUSTOMER-REGION
      MOVE 0 TO STATE-ERROR
      DISPLAY STATE-USPS-CODE (STATE-INDEX)
      " " STATE-INDEX WITH CONVERSION
      " " STATE-ERROR.
      SET STATE-INDEX UP BY 1.
      GO TO SEARCH-LOOP.

   SEARCH-END.

   The following lists the results of this serial search:
   IA 13 0 IL 15 0 IN 16 0 KS 17 0 MI 23 0 MN 24 0 NE 30 0 OH 36 0 WI 49 0

3. Serial search with two WHEN phrases:

   IA 13 0
   IL 15 0
**Procedure Division**

IN 16 0
KS 17 0
MI 23 0
MN 24 0
NE 30 0
OH 36 0
WI 49 0
INITIALIZE-SEARCH.
   MOVE 1 TO CUSTOMER-REGION.
   MOVE "NH" TO CUSTOMER-USPS-STATE.
   DISPLAY "States in customer's region:".

SEARCH-LOOP.
   SEARCH STATES
   AT END
   GO TO SEARCH-END
   WHEN STATE-USPS-CODE (STATE-INDEX) = CUSTOMER-USPS-STATE
   SET STATE-NUM TO STATE-INDEX
   WHEN STATE-REGION (STATE-INDEX) = CUSTOMER-REGION
   DISPLAY STATE-USPS-CODE (STATE-INDEX)
   " " WITH NO ADVANCING.
   SET STATE-INDEX UP BY 1.
   GO TO SEARCH-LOOP.

SEARCH-END.
   DISPLAY " "
   DISPLAY "Customer state index number = " STATE-NUM.

The following lists the results of the serial search with two WHEN phrases:

States in customer's region:
CT DC DE MA MD ME NJ NY PA RI VT

Customer state index number = 31

4. Updating a table in a SEARCH statement:

GET-NAME.
   DISPLAY "Enter name: " NO ADVANCING.
   ACCEPT CUSTOMER-NAME.
   SET NAME-INDEX TO 1.
   SEARCH NAME-ENTRY
   AT END
   DISPLAY " Table full"
   SET NAME-INDEX TO 1
   PERFORM SHOW-TABLE 8 TIMES
   STOP RUN
   WHEN LAST-NAME (NAME-INDEX) = CUSTOMER-NAME
   ADD 1 TO NAME-COUNT (NAME-INDEX)
   WHEN LAST-NAME (NAME-INDEX) = SPACES
   MOVE CUSTOMER-NAME TO LAST-NAME (NAME-INDEX)
   MOVE 1 TO NAME-COUNT (NAME-INDEX).
   GO TO GET-NAME.
SHOW-TABLE.
   DISPLAY LAST-NAME (NAME-INDEX) " " NAME-COUNT (NAME-INDEX).
   SET NAME-INDEX UP BY 1.

The following lists the results of updating a table in a SEARCH statement:
SET

Function

The SET statement sets values of indexes associated with table elements. It can also change the value of a conditional variable, change the status of an external switch, and store the address of a COBOL identifier reference at run time.

[result]

is an index-name, the identifier of an index data item, or an elementary numeric data item described as an integer.

[val]

is a positive integer, which can be signed. It can also be an index-name (or the identifier of an index data item) or an elementary numeric data item described as an integer.

[indx]

is an index-name.

[incem]

is an integer, which can be signed. It can also be the identifier of an elementary numeric data item described as an integer.

[cond-name]

is a condition-name that must be associated with a conditional variable.
[switch-name]
is the name of an external switch defined in the SPECIAL-NAMES paragraph.

[pointer-id]
is a data-name whose data description entry must contain the USAGE IS POINTER clause.

[identifier]
is a data item in the File, Working-Storage, Linkage, or Subschema Section.

[status-code-id]
is a word or longword integer data item represented by PIC S9(1) to S9(9) COMP or PIC 9(1) to 9(9) COMP.

**Syntax Rule**

No two occurrences of cond-name can refer to the same conditional variable.

**General Rules**

**Formats 1 and 2**

1. Index-names are associated with a table in the table's OCCURS clause INDEXED BY phrase.

2. If rsult is an index-name, its value after SET statement execution must correspond to an occurrence number of an element in the associated table.

3. If val is an index-name, its value before SET statement execution must correspond to an occurrence number of an element in the table associated with rsult.

4. The value of indx, both before and after SET statement execution, must correspond to an occurrence number of an element in the table associated with indx.

**Format 1**

5. The SET statement sets the value of rsult to refer to the table element whose occurrence number corresponds to the table element referred to by val. If val is an index data item, no conversion occurs.

6. If rsult is an index data item, val cannot be an integer. No conversion occurs when rsult is set to the value of val.

7. If rsult is not an index data item or an index-name, val can only be an index-name.

8. When there is more than one rsult, SET uses the original value of val in each operation. Subscript or index evaluation for rsult occurs immediately before its value changes.

9. Table 6.18, “Validity of Operand Combinations in Format 1 SET Statements” shows the validity of operand combinations. An asterisk (*) means that no conversion occurs during the SET operation.

**Table 6.18. Validity of Operand Combinations in Format 1 SET Statements**

<table>
<thead>
<tr>
<th>Sending Item</th>
<th>Integer Data Item</th>
<th>Receiving Item</th>
<th>Index Data Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer Literal</td>
<td>Invalid/Rule 7</td>
<td>Valid/Rule 5</td>
<td>Invalid/Rule 6</td>
</tr>
<tr>
<td>Integer Data Item</td>
<td>Invalid/Rule 7</td>
<td>Valid/Rule 5</td>
<td>Invalid/Rule 6</td>
</tr>
<tr>
<td>Index</td>
<td>Valid/Rule 7</td>
<td>Valid/Rule 5</td>
<td>Valid/Rule 6*</td>
</tr>
<tr>
<td>Sending Item</td>
<td>Receiving Item</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer Data Item</td>
<td>Index Data Item</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index Data Item</td>
<td>Invalid/Rule 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Valid/Rule 5*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Valid/Rule 6*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format 2**

10. The SET statement increments (UP) or decrements (DOWN) $indx$ by a value that corresponds to the number of occurrences $inrem$ represents.

11. When there is more than one $indx$, SET uses the original value of $inrem$ in each operation.

**Format 3**

12. SET moves the literal in the VALUE clause for $cond-name$ to its associated conditional variable. The transfer occurs according to the rules for elementary moves. If the VALUE clause contains more than one literal, the first is moved.

**Format 4**

13. SET changes the status of each $switch-name$ in the statement.

14. The ON phrase changes the status of $switch-name$ to on.

15. The OFF phrase changes the status of $switch-name$ to off.

16. The SET statement changes the switch status only for the image in which it executes. When the image terminates, the status of each external switch is the same as when the image began.

**Format 5**

17. The address of $identifier$ is evaluated and stored in $pointer-id$.

**Format 6**

18. Specifying the SUCCESS option sets $status-code-id$ to the SUCCESS state (the low-bit of $status-code-id$ is set to 1).

19. Specifying the FAILURE option sets $status-code-id$ to the FAILURE state (the low-bit of $status-code-id$ is set to 0).

**Additional References**

- SPECIAL-NAMES paragraph in Chapter 4, *Environment Division*
- Section 6.5.4: Switch-Status Condition
- Section 6.5.6: Success/Failure Condition
- MOVE statement
- PERFORM statement
- SEARCH statement

**Examples**

The examples assume these Environment and Data Division entries:

SPECIAL-NAMES.
SWITCH 1 UPDATE-RUN ON STATUS IS DO-UPDATE
SWITCH 3 REPORT-RUN ON STATUS IS DO-REPORT
OFF STATUS IS SKIP-REPORT
SWITCH 4 IS NEW-YEAR ON STATUS IS BEGIN-YEAR
OFF IS CONTINUE-YEAR.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 YEAR-LEVEL PIC 99.
   88 FRESHMAN VALUE 1.
   88 SOPHOMORE VALUE 2.
   88 JUNIOR VALUE 3.
   88 SENIOR VALUE 4.
   88 FIRST-MASTERS VALUE 5.
   88 MASTERS VALUE 5,6.
   88 FIRST-DOCTORAL VALUE 7.
   88 DOCTORAL VALUE 7,8.
   88 NON-DEGREE-UNDERGRAD VALUE 9.
   88 NON-DEGREE-GRAD VALUE 10.
   88 UNDERGRAD VALUE 9, 1 THROUGH 4.
   88 GRAD VALUE 10, 5 THROUGH 8.
01 COURSES-AVAILABLE.
   02 OCCURS 100 TIMES INDEXED BY COURSE-INDEX.
      03 COURSE-NAME PIC X(10).
      03 COURSE-INSTRUCTOR PIC X(20).
      03 COURSE-LOCATION PIC X(10).
      03 COURSE-CODE PIC 9(5).
01 POINTER-VAL USAGE IS POINTER.
01 THREE-DIMENSIONAL-TABLE.
   02 X OCCURS 5 TIMES INDEXED BY I.
      03 Y OCCURS 7 TIMES INDEXED BY J.
         04 Z PIC X(17) OCCURS 3 TIMES.
01 K PIC S9(9) COMP.
01 RETURN-STATUS PIC S9(9) COMP.
01 DECREMENT-VALUE PIC 9 VALUE 1.

1. Format 1—Initializing COURSE-INDEX.
   SET COURSE-INDEX TO 5.

2. Format 2—Adding to or subtracting from the index-name COURSE-INDEX.
   SET COURSE-INDEX UP BY 1.
   SET COURSE-INDEX DOWN BY DECREMENT-VALUE.

3. Format 3—Initializing a conditional variable:
   YEAR-LEVEL
   SET SOPHOMORE TO TRUE 02
   SET MASTERS TO TRUE 05
   SET GRAD TO TRUE 10
   SET NON-DEGREE-GRAD TO TRUE 10

4. Format 4—Setting external switches. The truth value shows the result of the IF statements:
   TRUTH VALUE
   SET UPDATE-RUN TO ON.
   SET REPORT-RUN TO OFF.
SET NEW-YEAR TO ON.
IF DO-UPDATE ...   true
IF DO-REPORT ...    false
IF CONTINUE-YEAR... false
SET REPORT-RUN TO ON.
IF DO-REPORT ...    true
IF SKIP-REPORT ...  false

5. Format 5—Setting POINTER-VAR to the address of the subscripted table item named Z(I,J,K).

SET POINTER-VAR TO REFERENCE OF Z(I,J,K).

6. Format 6—On OpenVMS Alpha and I64, initializing RETURN-STATUS to FAILURE before calling
subprogram SUBPROGA and a Run-Time Library Procedure, then checking for SUCCESS from each.

SET RETURN-STATUS TO FAILURE.
CALL "SUBPROGA" GIVING RETURN-STATUS.
IF RETURN-STATUS IS SUCCESS
   THEN
      GO TO A0200-Para
   ELSE
      DISPLAY "SUBPROGA failed"
      STOP RUN.
A0200-Para.
SET RETURN-STATUS TO FAILURE.
CALL "SCR$SET_CURSOR" USING BY VALUE 4, 22 GIVING RETURN-STATUS.
IF RETURN-STATUS IS SUCCESS
   THEN
      DISPLAY "UPDATE ROUTINE COMPLETED"
   ELSE
      DISPLAY "Cursor positioning failed"
      STOP RUN.

IDENTIFICATION DIVISION.
PROGRAM-ID. SUBPROGA.

01 PROGRAM-STATUS PIC S9(9) COMP.

PROCEDURE DIVISION GIVING PROGRAM-STATUS.
A000-BEGIN.

IF ... SET PROGRAM-STATUS TO SUCCESS
ELSE SET PROGRAM-STATUS TO FAILURE.
EXIT PROGRAM.
SORT

Function

The SORT statement (Format 1) creates a sort file by executing input procedures or transferring records from an input file. It sorts the records in the sort file using one or more keys that you specify. Finally, it returns each record from the sort file, in sorted order, to output procedures or an output file.

SORT (Format 2) orders the elements in a table. This is especially useful for tables used with SEARCH ALL. The table elements are sorted based on the keys as specified in the OCCURS for the table unless you override them by specifying keys in the SORT statement. If no key is specified, the table elements are the SORT keys.

[sortfile]

is a file-name described in a sort-merge file description (SD) entry in the Data Division.

[sortkey]

(Format 1) is the data-name of a data item in a record associated with sortfile.

(Format 2) is the data-name of a data item in the table-name table.

[first-proc]

is the section-name or paragraph-name of the first (or only) section or paragraph of the INPUT or OUTPUT procedure range.

[end-proc]

is the section-name or paragraph-name of the last section or paragraph of the INPUT or OUTPUT procedure range.

[infile]

is the file-name of the input file. It must be described in a file description (FD) entry in the Data Division.

[outfile]

is the file-name of the output file. It must be described in a file description (FD) entry in the Data Division.

[table-name (Alpha, I64)]

is a table described with OCCURS in the Data Division.

[alpha]

is an alphabet-name defined in the SPECIAL-NAMES paragraph of the Environment Division.

Syntax Rules

All Formats

1. You can use SORT statements anywhere in the Procedure Division except in:
   • Declaratives (Format 1)
   • SORT or MERGE statement input or output procedures
2. `sortkey` can be qualified.

3. `sortkey` cannot be in a group item that contains variable occurrence data items.

4. The `sortkey` description cannot contain an OCCURS clause or be subordinate to a data description entry that does.

**Format 1**

5. If `sortfile` contains variable-length records, `infile` records must not be smaller than the smallest in `sortfile` nor larger than the largest.

6. If `sortfile` contains fixed-length records, `infile` records must not be larger than the largest record described for `sortfile`.

7. If `outfile` contains variable-length records, `sortfile` records must not be smaller than the smallest in `outfile` nor larger than the largest.

8. If `outfile` contains fixed-length records, `sortfile` records must not be larger than the largest record described for `outfile`.

9. `sortfile` can have more than one record description. However, `sortkey` needs to be described in only one of the record descriptions. The character positions referenced by `sortkey` are used as the key for all the file's records.

10. The words THRU and THROUGH are equivalent.

11. If `outfile` is an indexed file, the first `sortkey` must be in the ASCENDING phrase. It must specify the same character positions in its record as the prime record key for `outfile`.

**Format 2 (Alpha, I64)**

12. `table-name` may be qualified and must have an OCCURS clause in its data description entry. If `table-name` is subject to more than one level of OCCURS clauses, subscripts must be specified for all levels with OCCURS INDEXED BY.

13. `table-name` is a key data-name, subject to the following rules:
   - The data item identified by a key data-name must be the same as, or subordinate to, the data item referenced by `table-name`.
   - Key data items may be qualified.
   - The data items identified by key data-names must not be variable-length data items.
   - If the data item identified by a key data-name is subordinate to `table-name`, it must not be described with an OCCURS clause, and it must not be subordinate to an entry that is also subordinate to `table-name` and contains an OCCURS clause.

14. The KEY phrase may be omitted only if the description of the table referenced by `table-name` contains a KEY phrase.

**General Rules**

**All Formats**

1. The first `sortkey` you specify is the major key, the next `sortkey` you specify is the next most significant key, and so forth. The significance of `sortkey` data items is not affected by how you divide them into KEY phrases. Only first-to-last order determines significance.
2. The ASCENDING phrase causes the sorted sequence to be from the lowest to highest sortkey value.

3. The DESCENDING phrase causes the sorted sequence to be from the highest to the lowest sortkey value.

4. Sort sequence follows the rules for relation condition comparisons.

5. The DUPLICATES phrase affects the return order of records or table elements whose corresponding sortkey values are equal.
   - When there is a USING phrase, return order is the same as the order of appearance of infile names in the SORT statement.
   - When there is an INPUT PROCEDURE, return order is the same as the order in which the records were released.
   - When table elements are returned, the order is the relative order of the contents of these table elements before sorting.

6. If there is no DUPLICATES phrase, the return order for records or table elements with equal corresponding sortkey values is unpredictable.

7. The SORT statement determines the comparison collating sequence for nonnumeric sortkey items when it begins execution. If there is a COLLATING SEQUENCE phrase in the SORT statement, SORT uses that sequence. Otherwise, it uses the program collating sequence described in the OBJECT-COMPUTER paragraph.

Format 1

8. If sortfile contains fixed-length records, any shorter infile records are space-filled on the right, following the last character. Space-filling occurs before the infile record is released to sortfile.

9. The INPUT PROCEDURE range consists of one or more sections or paragraphs that:
   - Appear contiguously in the source program
   - Do not form a part of an OUTPUT PROCEDURE range

10. The statements in the INPUT PROCEDURE range must include at least one RELEASE statement to transfer records to sortfile.

11. The INPUT PROCEDURE range can consist of any procedure needed to select, modify, or copy the next record made available by the RELEASE statement to the file referenced by sortfile.

12. The range of the INPUT PROCEDURE additionally includes all statements executed as a result of a CALL, EXIT, GO TO, or PERFORM statement. The range of the INPUT PROCEDURE also includes all statements in the Declaratives Section that can be executed if control is transferred from statements in the range of the INPUT PROCEDURE.

13. The INPUT PROCEDURE range must not contain MERGE, RETURN, or SORT statements.

14. If there is an INPUT PROCEDURE phrase, control transfers to the first statement in its range before the SORT statement sequences the sortfile records. When control passes the last statement in the INPUT PROCEDURE range, the records released to sortfile are sorted.

15. During execution of the INPUT or OUTPUT procedures, or any USE AFTER EXCEPTION procedure implicitly invoked during the SORT statement, no outside statement can manipulate the files or record areas associated with infile or outfile.

16. If there is a USING phrase, the SORT statement transfers all records in infile to sortfile. This transfer is an implied SORT statement input procedure. When the SORT statement executes, infile must not be open.

17. For each infile, the SORT statement:
• Initiates file processing as if the program had executed an OPEN statement with the INPUT phrase.

• Gets the logical records and releases them to the sort operation. SORT obtains each record as if the program had executed a READ statement with the NEXT and AT END phrases.

• Terminates file processing as if the program had executed a CLOSE statement with no optional phrases. The SORT statement ends file processing before it executes any output procedure.

These implicit OPEN, READ, and CLOSE operations cause associated USE procedures to execute when an exception condition occurs.

18. OUTPUT PROCEDURE consists of one or more sections or paragraphs that:

• Appear contiguously in the source program

• Do not form part of an INPUT PROCEDURE range

19. When the SORT statement begins the OUTPUT PROCEDURE phrase, it is ready to select the next record in sorted order. The statements in the OUTPUT PROCEDURE range must include at least one RETURN statement to make records available for processing.

20. When the MERGE statement enters the OUTPUT PROCEDURE range, it is ready to select the next record in merged order. Statements in the OUTPUT PROCEDURE range must execute at least one RETURN statement to make records available for processing.

21. The OUTPUT PROCEDURE can consist of any procedure needed to select, modify, or copy the next record made available by the RETURN statement in sorted order from the file referenced by sortfile.

22. The range of the OUTPUT PROCEDURE additionally includes all statements executed as a result of a CALL, EXIT, GO TO, or PERFORM statement. The range of the OUTPUT PROCEDURE also includes all statements in the Declarative USE procedures that can be executed if control is transferred from statements in the range of the OUTPUT PROCEDURE.

23. The OUTPUT PROCEDURE range must not include MERGE, RELEASE, or SORT statements.

24. If there is an OUTPUT PROCEDURE phrase, control passes to the first statement in its range after the SORT statement sequences the records in sortfile. When control passes the last statement in the OUTPUT PROCEDURE range, the SORT statement ends. Control then transfers to the next executable statement after the SORT statement.

25. If there is a GIVING phrase, the SORT statement writes all sorted records to each outfile. This transfer is an implied SORT output procedure. When the SORT statement executes, outfile must not be open.

26. The SORT statement initiates outfile processing as if the program had executed an OPEN statement with the OUTPUT phrase. The SORT statement does not initiate outfile processing until after INPUT PROCEDURE execution.

27. The SORT statement obtains the sorted logical records and writes them to each outfile. SORT writes each record as if the program had executed a WRITE statement with no optional phrases.

For relative files, the value of the relative key data item is 1 for the first returned record, 2 for the second, and so on. When the SORT statement ends, the value of the relative key data item indicates the number of outfile records.

28. The SORT statement terminates outfile processing as if the program had executed a CLOSE statement with no optional phrases.

29. These implicit OPEN, WRITE, and CLOSE operations can cause associated USE procedures to execute if they are present. If a USE procedure is present, processing terminates after the USE procedure has completed execution. If a USE procedure is not present, processing terminates as if the program had executed a CLOSE statement with no optional phrases.
30. If outfile contains fixed-length records, any shorter sortfile records are space-filled on the right, after the last character. Space-filling occurs before the sortfile record is released to outfile.

31. If the SORT statement is in a fixed segment, its input and output procedures must be completely in either:
   - Fixed segments
   - One independent segment

32. If the SORT statement is in an independent segment, its input and output procedures must be completely in either:
   - Fixed segments
   - The same independent segment as the SORT statement itself

**Format 2 (Alpha, I64)**

33. The SORT statement sorts the table referenced by table-name and presents the sorted table in table-name either in the order determined by the ASCENDING or DESCENDING phrases, if specified, or in the order determined by the KEY phrase associated with table-name.

34. To determine the relative order in which the table elements are stored after sorting, the contents of corresponding key data items are compared according to the rules for comparison of operands in a relation condition, starting with the most significant key data item.
   - If the contents of the corresponding key data items are not equal and the key is associated with the ASCENDING phrase, the table element containing the key data item with the lower value has the lower occurrence number.
   - If the contents of the corresponding key data items are not equal and the key is associated with the DESCENDING phrase, the table element containing the key data item with the higher value has the lower occurrence number.

**Additional References**

- OBJECT-COMPUTER paragraph in Chapter 4, Environment Division
- SPECIAL-NAMES paragraph in Chapter 4, Environment Division
- I-O-CONTROL paragraph in Chapter 4, Environment Division
- Section 6.5.1: Relation Conditions
- Section 6.7: Segmentation
- USE statement
- VSI COBOL User Manual, chapter on using SORT and MERGE statements

**Examples (Alpha, I64)**

The following examples all illustrate the use of table sorting (Format 2). For examples on Format 1 sorting, refer to the VSI COBOL User Manual.

The first example is a simple sort in which the table is sorted by the key definitions in the OCCURS clause of data item tbl. elem-item2 is the major key (ascending) and elem-item1 is the secondary key (descending). A SEARCH ALL statement is used.

```
identification division.
program-id. EXAMPLE1.
data division.
```
working-storage section.
01 group-item.
  05 tabl occurs 10 times
  ascending elem-item2
  descending elem-item1
  indexed by ind.
    10 elem-item1 pic x.
    10 elem-item2 pic x.
procedure division.
1. display "Example 1".
   move "13n3m3p3o3x1x1x1x1x1" to group-item.
   sort tabl.
   search all tabl at end
   display "not found"
   when elem-item1 (ind) = "m"
   if (elem-item1 (ind - 1) = "n")
      and (elem-item1 (ind + 1) = "1")
      display "elem-item1 is descending order - 2nd key"
   else
      display "elem-item1 is descending order - 2nd key"
   end-if
   end-search.
   exit program.
end program EXAMPLE1.

The following example is also a simple sort in which the table is sorted by the key definitions in the OCCURS clause of data item tabl. elem-item2 is the major key (ascending) and elem-item1 is the secondary key (descending).
A SEARCH ALL statement is used.

identification division.
program-id. EXAMPLE2.
data division.
working-storage section.
01 group-item.
  05 tabl occurs 10 times.
    10 elem-item1 pic x.
    10 elem-item2 pic x.
procedure division.
2. display "Example 2".
   move "13n3m3p3o3x1x1x1x1x1" to group-item.
   sort tabl ascending.
   if tabl (1) = "13"
      and tabl (2) = "m3"
      display "tabl is ascending order"
   else
      display "sort failed"
   end-if.
   exit program.
end program EXAMPLE2.

This following example is a simple sort in which the table is sorted in ascending order using each entire element of the table (data item tabl) to determine the sequence.

identification division.
program-id. EXAMPLE3.
data division.
working-storage section.
01 group-item.
05 tabl occurs 10 times
ascending elem-item3
descending elem-item1.
   10 elem-item1 pic x.
   10 elem-item2 pic x.
   10 elem-item3 pic x.

procedure division.
3. display "Example 3".
   move "l3bn3cm3ap3do3fx1ex1lx1hx1gx1a" to group-item.
   sort tabl descending elem-item2 elem-item3.
   if tabl (1) = "o3f"
      and tabl (2) = "p3d"
      display "tabl is descending order"
   else
      display "sort failed"
   end-if.
   exit program.
end program EXAMPLE3.

The following example sorts only the third instance of tabl2, that is, tabl1(3). The qualified data item, elem-item1 of group2 is its key. In normal PROCEDURE DIVISION reference, elem-item1 of group2 requires two levels of subscripting/indexing, whereas here it has none. Similarly, tabl2 normally requires one level of subscripting, but cannot be subscripted as data-name2 in the SORT statement. Instead it uses the value of t1-ind for determining which instance is sorted.

identification division.
program-id. EXAMPLE4.
data division.
working-storage section.
01 group-item.
   05 tabl1 occurs 3 times
      indexed by t1-ind t2-ind.
      10 tabl2 occurs 5 times.
         15 group1.
            20 elem-item1 pic x.
         15 group2.
            20 elem-item1 pic 9.

procedure division.
4. display "Example 4".
   move "x5z4y6z6x4a3b2b1a2c1j7j8k8l7j9" to group-item.
   set t1-ind to 3.
   sort tabl2 descending elem-item1 of group2.
   if group1 (3 1) = "j"
      and group2 (3 1) = "9"
      and tabl1 (1) = "x5z4y6z6x4"
      and tabl1 (2) = "a3b2b1a2c1"
      display "tabl1 (3) is descending order"
   else
      display "sort failed"
   end-if.
   exit program.
end program EXAMPLE4.
Function

The START statement establishes the logical position of the File Position Indicator in an indexed or relative file. The logical position affects subsequent sequential record retrieval.

[file-name]

is the name of an indexed or relative file with sequential or dynamic access. It cannot be the name of a sort or merge file.

[key-data]

is one of the following:

- The data-name specified as a record key
- The segmented-key name specified as a record key
- The leftmost part of a record key
- The relative key for [file-name]

It can be qualified.

[stmt]

is an imperative statement executed for an invalid key condition.

[stmt2]

is an imperative statement executed for a not invalid key condition.

Syntax Rules

1. To use the REGARDLESS or ALLOWING options, the program must include these entries:
   - APPLY LOCK-HOLDING clause of the I-O-CONTROL paragraph
   - ALLOWING option of the OPEN statement

2. There must be an INVALID KEY phrase if [file-name] does not have an applicable USE AFTER EXCEPTION procedure.

3. For a relative file, [key-data] must be the file's RELATIVE KEY data item.

4. For an indexed file, [key-data] can be either:
   - A record key for the file.
   - A data item subordinate to the description of a record key for the file. The data item must have the same leftmost character position as the record key, and must be one of the following:
     - A group, alphanumeric, or alphabetic item
     - An unsigned numeric display item
     - A COMP-3 integer or a COMP integer

   All the data types in the preceding list except alphanumeric are VSI extensions.
5. The REGARDLESS and ALLOWING options are VSI standard syntax, and cannot be used for a file connector that has had (on Alpha and I64) X/Open standard syntax (WITH [NO] LOCK or LOCK MODE) specified.

**General Rules**

### All Files

1. The file must be open in the INPUT or I-O mode when the START statement executes.

2. If there is no KEY phrase, the implied relational operator is EQUAL.

3. START statement execution does not change: (a) the contents of the record area or (b) the contents of the data item referred to in the DEPENDING ON phrase of the file’s RECORD clause.

4. The comparison specified by the KEY phrase relational operator occurs between a key for a record in the file and a data item. If the file is indexed, and the operand sizes are unequal, the comparison operates as if the longer one was truncated on the right to the size of the shorter.

5. START LESS can only be used with a file whose organization is INDEXED and whose access mode is DYNAMIC. The file must be opened for INPUT or I-O.

6. For indexed files, the file system compares the Key of Reference according to the native collating sequence and the sort order of the Key of Reference. The comparisons IS GREATER THAN, IS GREATER THAN OR EQUAL TO, and IS NOT LESS THAN refer to the logical record order, according to the sort order of the key. For example, if the sort order is descending, the KEY GREATER THAN key-data phrase positions the file at the next record whose key is less than key-data.

All other numeric or nonnumeric comparison rules apply.

The File Position Indicator is set to the first logical record in the file whose key satisfies the comparison.

If no record in the file satisfies the comparison:

- The invalid key condition exists.
- START statement execution is unsuccessful.
- The File Position Indicator denotes that no valid next record is established.

7. On Alpha and I64 systems, START LESS, LESS OR EQUAL, and NOT GREATER set the file position indicator by making reference to the logical record order in the same manner as START GREATER, GREATER OR EQUAL and NOT LESS.

8. The START verb can use the KEY IS syntax to establish the key field within the file record which is the Key of Reference. An immediately subsequent READ PRIOR will follow the order of the Key of Reference to access the logically previous record in the file according to that Key of Reference. If the KEY IS syntax is not used, the Key of Reference is understood to be the file’s primary key field.

9. On Alpha and I64 systems, when a successful START LESS, LESS OR EQUAL or NOT GREATER has occurred and the Key of Reference has ascending order, the record pointed to by the file position indicator can have the same key value or a smaller key value than the preceding record for the Key of Reference. If the Key of Reference has descending order, the record pointed to can have the same key value or a higher key value for the Key of Reference. The record pointed to can have the same key value if duplicate values for the Key of Reference exist on the file.

10. On Alpha and I64 systems, when an unsuccessful START LESS, LESS OR EQUAL or NOT GREATER has occurred the key of reference is undefined and a File Status value of 23 is returned, which indicates the INVALID KEY condition, or record not found.

11. The START statement updates the FILE STATUS data item for the file.
12. If the File Position Indicator denotes that an optional file is not present when the START statement executes, the invalid key condition exists. START statement execution is then unsuccessful.

13. The REGARDLESS and ALLOWING options can be used only in a manual record-locking environment. To create a manual record-locking environment, an access stream must specify the APPLY LOCK-HOLDING clause of the I-O-CONTROL paragraph.

14. The REGARDLESS option allows an access stream to position to a record regardless of any record locks held by other concurrent access streams. The START REGARDLESS option holds no lock on the record positioned to.

   This statement generates a soft record lock condition if the record that is pointed to is locked by another access stream. This condition results in a File Status value of 90 and invokes an applicable USE procedure, if any. Execution of the START REGARDLESS statement is considered successful and execution resumes at the next statement following the START REGARDLESS statement.

   However, on Tru64 UNIX systems, the soft lock condition (file status 90) is not recognized for indexed files. A START REGARDLESS statement for a record locked by another process performs the requested operation on the record and returns a file status of 00.

15. On OpenVMS, the ALLOWING UPDATERS option permits other concurrent access streams in the manual record-locking environment to simultaneously READ, DELETE, START, and REWRITE the current record. This option holds no lock on the current record.

16. The ALLOWING READERS option permits other concurrent access streams in the manual record-locking environment to simultaneously READ the current record. This option holds a read-lock on each such record read. No access stream can update the current record until it is unlocked.

17. On OpenVMS, the ALLOWING NO OTHERS option locks the current record. No other concurrent access stream can access this record until it is unlocked. Only this access stream can update this record.

18. On Tru64 UNIX systems, for indexed files the START statement (with or without the ALLOWING phrase) does not detect or acquire a record lock on the current record.

19. On Alpha and I64 systems, if X/Open file sharing is in effect, the START statement does not detect or acquire a lock.

20. If VSI standard record locking is in effect and the ALLOWING or REGARDLESS option is not specified, the default behavior for a START statement is that a lock is acquired if the file is opened in I-O mode and locks are detected in any mode.

21. On Alpha and I64, if ALLOWING or REGARDLESS is not specified, there is potential for ambiguity regarding VSI standard record locking or X/Open standard record locking. The selection of X/Open standard (rule 19) or VSI standard (rule 20) behavior is made as follows by the compiler:

   • If (on Alpha and I64) X/Open standard syntax (LOCK MODE or WITH (NO) LOCK) has been specified for file-name prior to the START statement, the compiler interprets the statement according to the X/Open standard.

   • If VSI standard syntax (LOCK-HOLDING, ALLOWING, or REGARDLESS) has been specified for file-name prior to the START statement, the compiler interprets the statement according to the VSI standard.

   • If no file-sharing syntax (LOCK-HOLDING, ALLOWING, REGARDLESS, LOCK MODE, or WITH [NO] LOCK) has been specified for file-name prior to the START statement, then the compiler uses the /STANDARD=[NO]XOPEN qualifier on OpenVMS Alpha and I64 (or the Tru64 UNIX equivalent -std [no]xopen flag) to determine whether the START statement is interpreted as X/Open or VSI standard: a setting of xopen selects the X/Open standard, whereas a setting of noxopen selects the VSI standard.

Any subsequent I-O locking syntax for the same file connector in your program must be consistent: X/Open standard locking (Alpha, I64) and VSI standard locking (implicit or explicit) cannot be mixed for the same file connector.
Relative Files

22. The comparison described in General Rule 4 uses the data item referred to by the RELATIVE KEY phrase in the file's ACCESS MODE clause.

Indexed Files

23. The START statement establishes a Key of Reference as follows:

- If there is no KEY phrase, the file's prime record key becomes the Key of Reference.
- If there is a KEY phrase, and key-data is a record key for the file, that record key becomes the Key of Reference.
- If there is a KEY phrase, and key-data is not a record key for the file, the record key whose leftmost character corresponds to the leftmost character of key-data becomes the Key of Reference.

The Key of Reference establishes the record ordering for the START statement. (See General Rule 4.) If the execution of the START statement is successful, later sequential READ statements use the same Key of Reference.

24. If there is a KEY phrase, the comparison described in General Rule 4 uses the contents of key-data.

25. If there is no KEY phrase, the comparison described in General Rule 4 uses the data item referred to in the file's RECORD KEY clause.

26. If START statement execution is not successful, the Key of Reference is undefined.

27. If there is an applicable USE AFTER EXCEPTION procedure, it executes whenever an input or output condition occurs that would result in a nonzero value in the first character of a FILE STATUS data item. However, it does not execute if the condition is invalid key and there is an INVALID KEY phrase.

See the rules for the INVALID KEY phrase, Section 6.6.10, “INVALID KEY Phrase”.

Technical Notes

- START statement execution can result in these FILE STATUS data item values:

<table>
<thead>
<tr>
<th>File Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Start is successful</td>
</tr>
<tr>
<td>23</td>
<td>Record not in file or optional file not present (invalid key)</td>
</tr>
<tr>
<td>47</td>
<td>File not open, or incompatible open mode</td>
</tr>
<tr>
<td>90</td>
<td>Record locked by another user; record available; soft lock</td>
</tr>
<tr>
<td>92</td>
<td>Record locked by another user; record not available; hard lock</td>
</tr>
<tr>
<td>30</td>
<td>All other permanent errors</td>
</tr>
</tbody>
</table>

Additional References

- LOCK MODE clause in the FILE-CONTROL paragraph in Chapter 4, Environment Division
- LOCK-HOLDING phrase in Section : APPLY Clause in I-O-CONTROL paragraph in Chapter 4, Environment Division
- Section 6.1.4: Scope of Statements
- Section 6.5.1.1: Comparison of Numeric Operands
Function

The STOP statement permanently terminates or temporarily suspends image execution.

[disp]

is any literal, or any figurative constant except ALL literal.

Syntax Rule

If a STOP RUN statement is in a consecutive sequence of imperative statements in a sentence, it must be the last statement in that sequence.

General Rules

1. STOP RUN terminates image execution.

2. STOP disp temporarily suspends the image. It displays the value of disp on the user's standard display device. If the user continues the image, execution resumes with the next executable statement.

Technical Notes

1. STOP RUN causes all open files to be closed before control returns to the operating system prompt.

2. STOP disp returns control to the operating system command language interpreter level without terminating the image as follows:
   • On Tru64 UNIX systems, STOP disp resumes execution when a carriage return is entered.
   • On Open VMS systems, control returns to DCL. The user can continue image execution with a CONTINUE command, which returns control to the program at the next executable statement.

Additional Reference (OpenVMS)

Function

The STRING statement concatenates the partial or complete contents of one or more data items into a single data item.

[src-string]
is a nonnumeric literal or identifier of a DISPLAY data item. It is the sending area.

[delim]
is a nonnumeric literal or the identifier of a DISPLAY data item. It is the delimiter of src-string.

[dest-string]
is the identifier of a DISPLAY data item. It cannot be reference modified. dest-string is the receiving area that contains the result of the concatenated src-strings.

[pointr]
is an elementary numeric data item described as an integer. It points to the position in dest-string to contain the next character moved.

[stmt]
is an imperative statement executed for an on overflow condition.

[stmt2]
is an imperative statement executed for a not on overflow condition.

Syntax Rules

1. pointr cannot define the assumed decimal scaling position character (P) in its PICTURE clause.
2. Literals can be any figurative constant other than ALL literal.
3. The description of dest-string cannot: (a) have a JUSTIFIED clause or (b) indicate an edited data item.
4. The size of pointr must allow it to contain a value one greater than the size of dest-string.

General Rules

1. delim specifies the characters to delimit the move.
2. If the size of delim is zero characters, it never matches a src-string delimiter.
3. If src-string is a variable-length item, SIZE refers to the number of characters currently defined for it.
4. When src-string or delim is a figurative constant, its size is one character.
5. The STRING statement moves characters from src-string to dest-string according to the rules for alphanumeric to alphanumeric moves. However, no space-filling occurs.
6. When the DELIMITED phrase contains delim:
   • The contents of each src-string are moved to dest-string in the sequence in which they appear in the statement.
   • Data movement begins with the leftmost character and continues to the right, character by character.
• Data movement ends when the STRING operation:
  a. Reaches the end of src-string
  b. Reaches the end of dest-string
  c. Detects the characters specified by delim

7. No data movement occurs if the size of src-string is zero characters.

8. When the DELIMITED phrase contains the SIZE phrase:
   • The contents of each src-string are moved to dest-string in the same sequence in which they appear in the statement.
   • Data movement begins with the leftmost character and continues to the right, character by character.
   • Data movement ends when the STRING operation either:
     a. Has transferred all data in each src-string
     b. Reaches the end of dest-string
   • If src-string is a variable-length data item, the STRING statement moves the number of characters currently defined for the data item.

9. When the POINTER phrase appears, the program must set pointr to an initial value greater than zero before executing the STRING statement.

10. When there is no POINTER phrase, the STRING statement operates as if pointr were set to an initial value of 1.

11. When the STRING statement transfers characters to dest-string, the moves operate as if:
   • The characters were moved one at a time from src-string.
   • Each character were moved to the position in dest-string indicated by pointr (if pointr does not exceed the length of dest-string).
   • The value of pointr were increased by one before moving the next character.

12. When the STRING statement ends, only those parts of dest-string referenced during statement execution change. The rest of dest-string contains the same data as before the STRING statement executed.

13. Before it moves each character to dest-string, the STRING statement tests the value of pointr.
    If pointr is less than 1 or greater than the number of character positions in dest-string, the STRING statement:
    • Moves no further data to dest-string
    • Executes the ON OVERFLOW phrase stment
    • Transfers control to the end of the STRING statement if there is no ON OVERFLOW phrase
    If pointr is not less than 1 or not greater than the number of character positions in dest-string after the data is transferred, the STRING statement:
    • Executes the NOT ON OVERFLOW phrase stment2 and then transfers control to the end of the STRING statement
    • Transfers control to the end of the STRING statement if the NOT ON OVERFLOW phrase is not specified

14. Subscript evaluation for dest-string and pointr occurs at the beginning of the statement.
### Additional References

- Section 6.1.4: Scope of Statements
- MOVE statement

### Examples

The examples assume the following data description entries:

```assembler
WORKING-STORAGE SECTION.
01 TEXT-STRING PIC X(30).
01 INPUT-MESSAGE PIC X(60).
01 NAME-ADDRESS-RECORD.
   03 CIVIL-TITLE PIC X(5).
   03 LAST-NAME PIC X(10).
   03 FIRST-NAME PIC X(10).
   03 STREET PIC X(15).
   03 CITY PIC X(15).
   * Assume CITY ends with "/"
   03 STATE PIC XX.
   03 ZIP PIC 9(5).
01 PTR PIC 99.
01 HOLD-PTR PIC 99.
01 LINE-COUNT PIC 99.
```

- Using both delimiters and SIZE:

```assembler
DISPLAY " ".
DISPLAY NAME-ADDRESS-RECORD.
MOVE SPACES TO TEXT-STRING.
STRING CIVIL-TITLE DELIMITED BY " " " " DELIMITED BY SIZE
   FIRST-NAME DELIMITED BY " " " " DELIMITED BY SIZE
   LAST-NAME DELIMITED BY SIZE
   INTO TEXT-STRING.
DISPLAY TEXT-STRING.
DISPLAY STREET.
MOVE SPACES TO TEXT-STRING.
STRING CITY DELIMITED BY "/" ", " DELIMITED BY SIZE
   STATE DELIMITED BY SIZE
   ZIP DELIMITED BY SIZE
   INTO TEXT-STRING.
DISPLAY TEXT-STRING.
```

### Results

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mr. Smith Irwin</td>
<td>603 Main St. Merrimack/</td>
<td>NH03054</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mr. Irwin Smith</td>
<td>603 Main St. Merrimack, NH 03054</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss Lambert Alice</td>
<td>1229 Exeter St. Boston/</td>
<td>MA03102</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss Alice Lambert</td>
<td>1229 Exeter St.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Boston, MA 03102
Mrs. Gilbert Rose 8 State Street New York/ NY10002
Mrs. Rose Gilbert 8 State Street New York, NY 10002
Mr. Cowherd Owen 1064 A St. Washington/ DC20002
Mr. Owen Cowherd 1064 A St. Washington, DC 20002

• Using the PROCESS phrase:

MOVE 0 TO LINE-COUNT.
MOVE 1 TO PTR.
GET-WORD.
IF LINE-COUNT NOT < 4
   DISPLAY " " TEXT-STRING
   GO TO GOT-WORDS.
   ACCEPT INPUT-MESSAGE.
   DISPLAY INPUT-MESSAGE.
SAME-WORD.
   MOVE PTR TO HOLD-PTR.
   STRING INPUT-MESSAGE DELIMITED BY SPACE ", " DELIMITED BY SIZE INTO TEXT-STRING WITH POINTER PTR ON OVERFLOW
   STRING " " DELIMITED BY SIZE INTO TEXT-STRING
   WITH POINTER HOLD-PTR
   DISPLAY " " TEXT-STRING
   MOVE SPACES TO TEXT-STRING
   ADD 1 TO LINE-COUNT
   MOVE 1 TO PTR
   GO TO SAME-WORD.
   GO TO GET-WORD.
GOT-WORDS.
EXIT.

Results
This example demonstrates how the STRING statement can construct text strings
that can construct text, using

377
the
POINTER
phrase
strings, using, the, POINTER,
phrase,

SUBTRACT

SUBTRACT

Function

The SUBTRACT statement subtracts one, or the sum of two or more, numeric items from one or more items. It stores the difference in one or more items.

[num]

is a numeric literal or the identifier of an elementary numeric item.

[result]

is the identifier of an elementary numeric item. However, in Format 2, result can be an elementary numeric edited item. It is the resultant identifier.

[stmt]

is an imperative statement executed when a size error condition has occurred.

[stmt2]

is an imperative statement executed when no size error condition has occurred.

[grp-1]

is the identifier of a group item.

[grp-2]

is the identifier of a group item.

Syntax Rule

CORR is an abbreviation for CORRESPONDING.

General Rules

1. In Format 1, the values of the operands before the word FROM are summed. This total is then subtracted from each result.

2. In Format 2, the values of the operands before the word FROM are summed. This total is subtracted from the num following the word FROM. The result replaces the current value of each result.

3. In Format 3, data items in grp-1 are subtracted from and stored in the corresponding data items in grp-2.

Additional References

• Section 6.1.4: Scope of Statements
Procedure Division

• Section 6.6.1: Arithmetic Operations
• Section 6.6.2: Multiple Receiving Fields in Arithmetic Statements
• Section 6.6.3: ROUNDED Phrase
• Section 6.6.4: ON SIZE ERROR Phrase
• Section 6.6.5: CORRESPONDING Phrase
• Section 6.6.7: Overlapping Operands and Incompatible Data

Examples

Each of the examples assume these data descriptions and initial values.

INITIAL VALUES

03 ITEMA PIC S99 VALUE -85. -85
03 ITEMB PIC 99 VALUE 2. 2
03 ITEMC VALUE "123".
05 ITEMD OCCURS 3 TIMES 1 2 3
   PIC 9.
03 ITEME PIC S99 VALUE -95. -95

1. Without GIVING phrase: RESULTS

   SUBTRACT 2 ITEMB FROM ITEMA. ITEMA = -89

2. SIZE ERROR clause:

   (When the size error condition occurs and the SIZE ERROR clause is specified, the values of the affected
   resultant identifiers do not change.)

   SUBTRACT 14 FROM ITEMA, ITEME
   ON SIZE ERROR
   MOVE 0 TO ITEMB.
   ITEMA = -99
   ITEME = -95
   ITEMB = 0

3. NOT ON SIZE ERROR clause:

   SUBTRACT 14 FROM ITEMA
   ON SIZE ERROR
   MOVE 9 TO ITEMB.
   NOT ON SIZE ERROR
   MOVE 1 TO ITEMB.
   ITEMA = -99
   ITEMB = 1

4. Multiple receiving fields:

   (The operations proceed from left to right. Therefore, the subscript for ITEMB is evaluated after the subtraction
   changes its value.)

   SUBTRACT 1 FROM ITEMB ITEMD (ITEMB).
   ITEMB = 1
   ITEMD (1) = 0

5. GIVING phrase:

   SUBTRACT ITEME ITEMD (ITEMB) FROM ITEMA
   GIVING ITEMB.
   ITEMB = 8

6. END-SUBTRACT:
(The first SUBTRACT terminates with END-SUBTRACT. If the SIZE ERROR condition had not occurred, the second SUBTRACT statement would have executed anyway: the value of ITEMA would have been -86.)

```plaintext
SUBTRACT 10 ITEMB FROM ITEMD (ITEMB)         ITEMD (2) = 2
   ON SIZE ERROR                              ITEMA = 0
      MOVE 0 TO ITEMA
   END-SUBTRACT.
SUBTRACT 1 FROM ITEMA.                     ITEMA = -1
```

(The following example shows the usefulness of END-SUBTRACT inside an IF statement. Without it, there would be no way to code the DISPLAY statements.)

```plaintext
IF ITEMB < 3 AND > 1
   SUBTRACT 1 FROM ITEMD (ITEMB)
   ON SIZE ERROR
      MOVE 0 TO ITEMA
   END-SUBTRACT
   DISPLAY 'yes'
ELSE
   DISPLAY 'no'.
```

**SUPPRESS**

**Function**

The SUPPRESS statement causes the Report Writer Control System (RWCS) to inhibit the presentation of a report group.

**Syntax Rule**

The SUPPRESS statement can appear only in a USE BEFORE REPORTING Declarative procedure.

**General Rules**

1. The SUPPRESS statement inhibits only the presentation of a report-group-name (a 01-level Report Group Description entry).

2. Each time the presentation of a report group is to be inhibited, the program must execute a SUPPRESS statement.

3. The SUPPRESS statement directs the Report Writer Control System (RWCS) to inhibit the processing of these report group functions:
   - The presentation of the print lines
   - The processing of all LINE clauses
   - The processing of the NEXT GROUP clause
   - The adjustment of LINE-COUNTER

4. The SUPPRESS statement does not inhibit the processing of sum counters or control breaks.
Additional References

- Section 6.6.7: Overlapping Operands and Incompatible Data
- USE statement

Example

PROCEDURE DIVISION.
DECLARATIVES.
DET SECTION.
  USE BEFORE REPORTING DETAIL-LINE.
DETA-1.
  IF SORTED-NAME = NAME
    ADD A TO B
    SUPPRESS PRINTING.
  IF NAME = SPACES SUPPRESS PRINTING.
END DECLARATIVES.
MAIN SECTION.
  .
  .

TERMINATE

TERMINATE

Function

The TERMINATE statement causes the Report Writer Control System (RWCS) to complete the processing of the specified report.

[report-name]

names a report defined by a Report Description entry in the Report Section of the Data Division.

General Rules

1. If the TERMINATE statement includes more than one report-name, the statement executes as if there were a separate TERMINATE statement for each report-name.

2. The program cannot execute a TERMINATE statement unless an INITIATE statement was executed before the TERMINATE statement for that report, and the program did not already execute a TERMINATE statement for that report.

3. If the program did not execute a GENERATE statement, the execution of a TERMINATE statement does not cause the RWCS to produce any of its report groups or perform any of the related processing.

4. The TERMINATE statement causes the RWCS to:
   - Produce all CONTROL FOOTING report groups beginning with the minor CONTROL FOOTING report group.
   - Produce the REPORT FOOTING report group.

The RWCS makes the prior set of control data item values available to these two report groups and to any associated USE procedure. This action simulates a control break at the most major level.
5. The RWCS automatically processes the PAGE HEADING and PAGE FOOTING report groups, if present, when it must advance the report to a new page to present a CONTROL HEADING, DETAIL, or CONTROL FOOTING report group.

6. The TERMINATE statement does not automatically close a report file; the program must close the file. The program must terminate the report before the CLOSE statement can close the report file.

**Additional Reference**

USE statement.

**UNLOCK**

**UNLOCK**

**Function**

The UNLOCK statement removes a record lock from the current record or from all locked records in the file. On Alpha and I64 systems, the X/Open standard UNLOCK statement always removes the record lock from all locked records in the file.

```
[file-name]
```

is the name of a sequential, relative, or indexed file described in the Data Division.

**Syntax Rules**

1. For Format 1, if the UNLOCK statement does not include the RECORD or the ALL RECORDS option, the singular RECORD option is the default. (However, see General Rule 3.)

2. For Format 2, the RECORD and RECORDS options have the same effect: to unlock all currently locked records. This behavior also is the default if neither option is specified.

**General Rules**

1. The first access stream to lock a record owns the record lock for that record.

2. Only the owner of a record lock can unlock the record.

3. For Format 1, implicitly (by default) or explicitly specifying the RECORD option unlocks the current record. Therefore, you must specify ALL RECORDS explicitly to unlock all the record locks held on `file-name`.

   The single exception to this rule for Format 1 is that for indexed files the RECORD option (implicitly or explicitly) is unsupported on Tru64 UNIX systems. The ALL RECORDS phrase is assumed.

4. For Format 2, whether you specify the RECORD option or the RECORDS option, the effect is the same: to unlock all record locks held on `file-name` by the current access stream.

5. If an access stream attempts to unlock a record (or records) in a file containing no record locks, the statement is considered successful and execution resumes at the statement following the UNLOCK statement.

6. Because both formats of the UNLOCK statement include the UNLOCK RECORD and UNLOCK forms, the compiler determines whether to interpret these forms of the statement as X/Open standard (on Alpha and I64) or VSI standard as follows:

   - If on Alpha and I64 X/Open standard syntax (LOCK MODE or WITH (NO) LOCK) has been specified for `file-name` prior to the UNLOCK statement, the compiler interprets the statement according to the X/Open standard.
• If VSI standard syntax (LOCK-HOLDING, ALLOWING, or REGARDLESS) has been specified for \textit{file-name} prior to the UNLOCK statement, the compiler interprets the statement according to the VSI standard.

• If no file-sharing syntax (LOCK-HOLDING, ALLOWING, REGARDLESS, LOCK MODE, or WITH [NO] LOCK) has been specified for \textit{file-name} prior to the UNLOCK statement, then the compiler uses the /STANDARD=[NO]XOPEN qualifier on OpenVMS Alpha and I64 (or the Tru64 UNIX equivalent -std [no]xopen flag) to determine whether the START statement is interpreted as X/Open or VSI standard: a setting of \texttt{xopen} selects the X/Open standard, whereas a setting of \texttt{noxopen} selects the VSI standard.

Any subsequent I-O locking syntax for the same file connector in your program must be consistent: X/Open standard locking and VSI standard locking (implicit or explicit) cannot be mixed for the same file connector.

**Technical Notes**

• UNLOCK statement execution can result in these FILE STATUS data item values:

<table>
<thead>
<tr>
<th>File Status</th>
<th>File Organization</th>
<th>Access Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>All</td>
<td>All</td>
<td>Unlock is successful</td>
</tr>
<tr>
<td>93</td>
<td>All</td>
<td>All</td>
<td>No current record</td>
</tr>
<tr>
<td>94</td>
<td>All</td>
<td>All</td>
<td>File not open, or incompatible open mode</td>
</tr>
<tr>
<td>30</td>
<td>All</td>
<td>All</td>
<td>All other permanent errors</td>
</tr>
</tbody>
</table>

**Additional References**

• LOCK-HOLDING phrase in Section: APPLY Clause in I-O-CONTROL paragraph in Chapter 4, \textit{Environment Division}

• LOCK MODE phrase in SELECT clause in File Control paragraph in Chapter 4, \textit{Environment Division}

• Technical Notes for DELETE statement

• OPEN statement

**VSI Standard Examples**

These examples assume only one access stream for the image. The following examples refer to this partial program:

```
CONFIGURATION SECTION.
FILE-CONTROL.
   SELECT MASTER-FILE ASSIGN TO "CLIENT.DAT"
   ORGANIZATION IS INDEXED
   ACCESS MODE IS DYNAMIC
   RECORD KEY IS MASTER-KEY
   FILE STATUS IS FILE-STAT.
I-O-CONTROL.

* * This APPLY clause is required syntax for manual record locking
* *

   APPLY LOCK-HOLDING ON MASTER-FILE.
```

```
DATA DIVISION.
FD MASTER-FILE
   LABEL RECORDS STANDARD.
```
01 MASTER-RECORD.
  
  PROCEDURE DIVISION.
A100-BEGIN.

  *  The ALLOWING phrase enables file sharing
  *

    OPEN I-O MASTER-FILE ALLOWING ALL.

A900-END-OF-JOB.

1. Unlocking the record lock on the current record by taking the default RECORD option:

    READ MASTER-FILE KEY IS MASTER-KEY
    ALLOWING NO OTHERS.
    REWRITE MASTER-RECORD ALLOWING NO OTHERS.
    UNLOCK MASTER-FILE.

2. Explicitly unlocking the record lock on the current record:

    READ MASTER-FILE KEY IS MASTER-KEY
    ALLOWING NO OTHERS.
    
    UNLOCK MASTER-FILE RECORD.

3. Unlocking all records in MASTER-FILE:

    PERFORM A100-READ-MASTER UNTIL
    MASTER-KEY = ID-KEY
    OR
    MASTER-KEY > ID-KEY.
    
    UNLOCK MASTER-FILE ALL RECORDS.

A100-READ-MASTER.
    READ MASTER-FILE ALLOWING NO OTHERS.

X/Open Standard Example (Alpha, I64)

The following example shows the use of X/Open standard syntax:

    SELECT employee-file ASSIGN TO "EMPFIL"
    ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS employee-id
    LOCK MANUAL LOCK ON MULTIPLE RECORDS
    FILE STATUS IS emp-stat.
* The file is implicitly shareable via the SELECT specification.

OPEN I-O employee-file.

PERFORM UNTIL emp-stat = end-of-file
    READ employee-file NEXT RECORD
    WITH LOCK
    IF employee-job-code = peon-code
        PERFORM find-boss-record
        ENDIF
REWRITE employee-record

* This will unlock this record and the boss's record found earlier.

UNLOCK employee-file RECORDS

END-PERFORM.

FIND-BOSS-RECORD.
    START employee-file
    KEY > employee-job-code.
    READ employee-file NEXT WITH LOCK.

UNSTRING

UNSTRING

Function

The UNSTRING statement separates contiguous data in a sending field and stores it in one or more receiving fields.

[src-string]

is the identifier of an alphanumeric class data item. It cannot be reference modified. Src-string is the sending field.

[delim]

is a nonnumeric literal or the identifier of an alphanumeric data item. It is the delimiter for the UNSTRING operation.

[dest-string]

is the identifier of an alphanumeric, alphabetic, or numeric DISPLAY data item. It is the receiving field for the data from src-string.

[delim-dest]

is the identifier of an alphanumeric data item. It is the receiving field for delimiters.
is the identifier of an elementary numeric data item described as an integer. It contains the count of characters moved.

[pointr]

is the identifier of an elementary numeric data item described as an integer. It points to the current character position in src-string.

[tally-ctr]

is the identifier of an elementary numeric data item described as an integer. It counts the number of dest-string fields accessed during the UNSTRING operation.

[stmt]

is an imperative statement executed for an on overflow condition.

[stmt2]

is an imperative statement executed for a not on overflow condition.

**Syntax Rules**

1. Literals can be any figurative constant other than ALL literal.

2. pointr must be large enough to contain a value one greater than the size of src-string.

3. The DELIMITER IN and COUNT IN phrases can appear only if there is a DELIMITED BY phrase.

4. countr, pointr, dest-string, and tally-ctr cannot define the assumed decimal scaling position character P in their PICTURE clauses.

**General Rules**

1. countr represents the number of characters in src-string isolated by the delimiters for the move to dest-string. The count does not include the delimiter characters.

2. When delim is a figurative constant, its length is one character.

3. When the ALL phrase is present:
   - One occurrence, or two or more contiguous occurrences, of delim (whether or not they are figurative constants) is treated as only one occurrence.
   - One occurrence of delim is moved to delim-dest when there is a DELIMITER IN phrase.

4. When any examination finds two contiguous delimiters, the current dest-string is filled with:
   - Spaces, if its class is alphabetic or alphanumeric
   - Zeros, if its class is numeric

5. delim can contain any characters in the computer character set.

6. Each delim is one delimiter. When delim contains more than one character, all its characters must be in src-string (in contiguous positions and the given order) to qualify as a delimiter.

7. When the DELIMITED BY phrase contains an OR phrase, an OR condition exists between all occurrences of delim. Each delim is compared to src-string. If a match occurs, the character in src-string is a single delimiter. No character in src-string can be part of more than one delimiter.
8. Each `delim` applies to `src-string` in the order it appears in the UNSTRING statement.

9. When execution of the UNSTRING statement begins, the current receiving field is the first `dest-string`.

10. If there is a POINTER phrase, the string of characters in `src-string` is examined, beginning with the position indicated by `pointr`. Otherwise, examination begins with the leftmost character position.

11. If there is a DELIMITED BY phrase, examination proceeds to the right until the UNSTRING statement detects `delim`. (See General Rule 6.)

12. If there is no DELIMITED BY phrase, the number of characters examined equals the size of the current `dest-string`. However, if the sign of `dest-string` is defined as occupying a separate character position, UNSTRING examines one less character than the size of `dest-string`. If `dest-string` is a variable-length data item, its current size determines the number of characters examined.

13. If the UNSTRING statement reaches the end of `src-string` before detecting the delimiting condition, examination ends with the last character examined.

14. The characters examined (excluding `delim`) are:
   - Treated as an elementary alphanumeric data item
   - Moved to the current `dest-string` according to the MOVE statement rules

15. When there is a DELIMITER IN phrase, the delimiter is:
   - Treated as an elementary alphanumeric data item
   - Moved to `delim-dest` according to the MOVE statement rules
   If the delimiting condition is the end of `src-string`, `delim-dest` is space-filled.

16. The COUNT IN phrase causes the UNSTRING statement to:
   - Count the number of characters examined (excluding the delimiter).
   - Move the count to `countr` according to the elementary move rules.

17. When there is a DELIMITED BY phrase, UNSTRING continues examining characters immediately to the right of the delimiter. Otherwise, examination continues with the character immediately to the right of the last one transferred.

18. After data transfers to `dest-string`, the next `dest-string` becomes the current receiving field.

19. The process described in General Rules 12 to 18 repeats until either:
   - There are no more characters in `src-string`.
   - The last `dest-string` has been processed.

20. The UNSTRING statement does not initialize `pointr` or `tally-ctr`. The program must set their initial values before executing the UNSTRING statement.

21. The UNSTRING statement adds one to `pointr` for each character it examines in `src-string`. When UNSTRING execution ends, `pointr` contains a value equal to its beginning value plus the number of characters the statement examined in `src-string`.

22. At the end of an UNSTRING statement with the TALLYING phrase, `tally-ctr` contains a value equal to its beginning value plus the number of `dest-string` fields the statement accessed.

23. An overflow condition can arise from either of these conditions:
• When the UNSTRING statement begins, the value of pointr is less than one or greater than the number of characters in src-string.

• During UNSTRING execution, all dest-string fields have been processed, and there are unexamined src-string characters.

24. When an overflow condition occurs, if there is a NOT ON OVERFLOW phrase, this phrase is ignored and the UNSTRING operation ends. If there is an ON OVERFLOW phrase, stment executes. Otherwise, control passes to the end of the UNSTRING statement.

25. At the end of the UNSTRING operation, when an overflow condition does not exist, the ON OVERFLOW phrase is ignored and the UNSTRING operation ends if a NOT ON OVERFLOW phrase does not exist. If there is a NOT ON OVERFLOW phrase, stment2 executes. After stment2 executes, control is passed to the end of the UNSTRING statement.

26. If there is a DELIMITED BY phrase and the size of dest-string is zero characters, no characters are moved. However, delim-dest contains the matched delimiter and countr contains the character count.

27. If there is no DELIMITED BY phrase and the size of dest-string is zero characters, no characters are moved. The value of pointr does not change. UNSTRING continues with the next dest-string.

28. If the size of delim is zero characters, delim does not match any characters in src-string.

Additional References

• Section 6.1.4: Scope of Statements
• MOVE statement

Examples

The examples assume these data descriptions:

WORKING-STORAGE SECTION.
01 INMESSAGE PIC X(20).
01 THEDATE.
   03 THEYEAR PIC XX JUST RIGHT.
   03 THEMONTH PIC XX JUST RIGHT.
   03 THEDAY PIC XX JUST RIGHT.
01 HOLD-DELIM PIC XX.
01 PTR PIC 99.
01 FIELD-COUNT PIC 99.
01 MONTH-COUNT PIC 99.
01 DAY-COUNT PIC 99.
01 YEAR-COUNT PIC 99.

• With OVERFLOW phrase:

DISPLAY "Enter a date: " NO ADVANCING.
ACCEPT INMESSAGE.
UNSTRING INMESSAGE
   DELIMITED BY "-" OR "/" OR ALL " "
       INTO THEMONTH DELIMITER IN HOLD-DELIM
       THEDAY DELIMITER IN HOLD-DELIM
       THEYEAR DELIMITER IN HOLD-DELIM
   ON OVERFLOW MOVE ALL "0" TO THEDATE.
   INSPECT THEDATE REPLACING ALL " " BY "0".
DISPLAY THEDATE.
Results

Enter a date: 6/13/87
870613
Enter a date: 6-13-87
870613
Enter a date: 6-13 87
870613
Enter a date: 6/13/87/2
000000
Enter a date: 1-2-3
030102

• With POINTER and TALLYING phrases:

DISPLAY "Enter two dates in a row: " NO ADVANCING.
ACCEPT INMESSAGE.
MOVE 1 TO PTR.
PERFORM DISPLAY-TWO 2 TIMES.
GO TO DISPLAYED-TWO.
DISPLAY-TWO.
MOVE SPACES TO THEDATE.
MOVE 0 TO FIELD-COUNT.
UNSTRING INMESSAGE
DELIMITED BY "-" OR "/" OR ALL " "
INTO THEMONTH DELIMITER IN HOLD-DELIM
THEDAY DELIMITER IN HOLD-DELIM
THEYEAR DELIMITER IN HOLD-DELIM
WITH POINTER PTR
TALLYING IN FIELD-COUNT.
INSPECT THEDATE REPLACING ALL " " BY "0".
DISPLAY THEDATE " " PTR " " FIELD-COUNT.
DISPLAYED-TWO.
EXIT.

Results

Enter two dates in a row: 6/13/87 8/15/87
870613 09 03
870815 21 03
Enter two dates in a row: 10 15 87-1 1 88
871015 10 03
880101 21 03
Enter two dates in a row: 6/13/87-12/31/87
870613 09 03
871231 21 03
Enter two dates in a row: 6/13/87-12/31
870613 09 03
001231 21 02
Enter two dates in a row: 6/13/87/12/31/87
870613 09 03
871231 21 03

• With COUNT phrase:

DISPLAY "Enter two dates in a row: " NO ADVANCING.
ACCEPT INMESSAGE.
MOVE 1 TO PTR.
PERFORM DISPLAY-TWO 2 TIMES.
GO TO DISPLAYED-TWO.

DISPLAY-TWO.
MOVE SPACES TO THEDATE.
MOVE 0 TO FIELD-COUNT MONTH-COUNT DAY-COUNT YEAR-COUNT.
UNSTRING INMESSAGE
   DELIMITED BY "-" OR "/" OR ALL " 
   INTO THEMONTH DELIMITER IN HOLD-DELIM COUNT MONTH-COUNT
   THEDAY DELIMITER IN HOLD-DELIM COUNT DAY-COUNT
   THEYEAR DELIMITER IN HOLD-DELIM COUNT YEAR-COUNT
   WITH POINTER PTR
   TALLYING IN FIELD-COUNT.
INSPECT THEDATE REPLACING ALL " " BY "0".
DISPLAY THEDATE " " PTR " " FIELD-COUNT
" : " MONTH-COUNT "-" DAY-COUNT "-" YEAR-COUNT.
DISPLAYED-TWO.
EXIT.

Results

Enter two dates in a row: 6/13/87 8/15/87
870613  09  03 : 01-02-02
870815  21  03 : 01-02-02
Enter two dates in a row: 10 15 87-1 1 88
871015  10  03 : 02-02-02
880101  21  03 : 01-01-02
Enter two dates in a row: 6/13/87-12/31/87
870613  09  03 : 01-02-02
871231  21  03 : 02-02-02
Enter two dates in a row: 6/13/87-12/31
870613  09  03 : 01-02-02
001231  21  02 : 02-02-00
Enter two dates in a row: 6/13/87/12/31/87
870613  09  03 : 01-02-02
871231  21  02 : 02-02-02

USE

USE

Function

The USE statement specifies Declarative USE procedures to handle input/output exceptions and errors. It can also specify procedures to be executed before the program processes a specific report group.

[file-name]

is the name of a file connector described in a file description entry in a Data Division. It cannot refer to a sort or merge file.

[group-data-name]

is the name of a report group in a report group description entry in a Data Division. It must not appear in more than one USE statement.
Syntax Rules

All Formats

1. A USE statement can be used only in a sentence immediately after a section header in the Procedure Division declaratives area. It must be the only statement in the sentence. The rest of the section can contain zero, one, or more paragraphs to define the USE procedures.

2. The USE statement itself does not execute. It defines the conditions that cause execution of the USE procedure.

Format 1

3. The ERROR and EXCEPTION syntax are equivalent and interchangeable.

Format 2

4. Of the four Report Writer Procedure Division verbs (SUPPRESS, GENERATE, INITIATE, or TERMINATE), only the SUPPRESS statement can appear in a USE BEFORE REPORTING procedure. A PERFORM statement in a USE BEFORE REPORTING procedure must not have GENERATE, INITIATE, or TERMINATE statements in its range.

The USE procedure must not alter the value of any control data item.

General Rules

All Formats

1. At run time, two special precedence rules apply for the selection of a USE procedure when a program is contained within another program. In applying these rules, only the first qualifying USE procedure is selected for execution. The order of precedence for the selection of a USE procedure is as follows:

   • First, select the applicable USE procedure within the program containing the statement that caused the qualifying condition.

   • If a USE procedure is not found in the program using the previous rule, the Run-Time System searches all programs directly or indirectly containing that program for a USE GLOBAL procedure. This search continues until the Run-Time System either: (a) finds an applicable USE GLOBAL procedure, or (b) finds the outermost containing program, if there is no applicable USE GLOBAL procedure. Either condition terminates the search.

2. A Declarative USE procedure cannot refer to a non-Declarative procedure. However, only the PERFORM statement can transfer execution control from:

   • A Declarative USE procedure to another Declarative USE procedure

   • A non-Declarative procedure to a Declarative USE procedure

3. After a USE procedure executes, control returns to the next executable statement in the invoking routine, if one is defined. Otherwise, control transfers according to the rules for Explicit and Implicit Transfers of Control.

4. A program must not execute a statement in a USE procedure that would cause execution of a USE procedure that had been previously executed and had not yet returned control to the routine that invoked it.

Format 1

5. A USE procedure executes automatically:
• After the system's input-output exception processing completes

• When an invalid key or at end condition results from an input-output statement that has no INVALID KEY or AT END clause

6. If there is an applicable USE AFTER EXCEPTION procedure, it executes whenever an input or output condition occurs that would result in a nonzero value in the first character of a FILE STATUS data item. However, it does not execute if: (a) the condition is invalid key and there is an INVALID KEY phrase, or (b) the condition is at end, and there is an AT END phrase.

7. One input-output exception cannot cause more than one USE AFTER EXCEPTION procedure to execute.

8. More than one USE AFTER EXCEPTION procedure can relate to an input-output operation when there is one procedure for file-name and another for the applicable open mode. In this case, only the procedure for file-name executes. This rule applies only to USE procedures in the same program.

9. If no applicable USE procedures are found in the local program, then containing programs are searched upwards for: (a) USE GLOBAL procedures for the file, and then (b) for USE GLOBAL procedures for the input-output type.

10. A USE AFTER EXCEPTION procedure specifying an open mode applies to an input-output operation only when all of the following are true:

  • The open mode (INPUT, OUTPUT, I-O, or EXTEND) specified in the USE AFTER EXCEPTION procedure is identical to the open mode in effect (that is, the open mode established by the OPEN statement).

  • The file is open or in the process of being opened.

  • There is no file-name declarative procedure for that file within the same program.

11. If an input-output error occurs for a file that is not open or not in the process of being opened, the only applicable USE procedure is a file-name USE procedure.

Format 2

12. The Report Writer Control System (RWCS) executes the USE BEFORE REPORTING procedure before it processes the named group-data-name report group. Only during the processing of the report group does the RWCS change prior values, execute control breaks, adjust LINE-COUNTER and PAGE-COUNTER, and present the report group.

Additional References

• Section 6.2.6: Scope of Names

• Section 6.3.1: Explicit and Implicit Procedure Division References

• Description of exception handling in the VSI COBOL User Manual.

Example

**************************************************************
* This example assumes that SELECT and FD statements exist*
* for FILE1-SEQ, FILE1-RAN, FILE1-DYN and FILE1-EXT.*
* All three USE procedures are local to the program*
* that hosts this fragment.
* At run-time if there is an exception on opening FILE1-RAN*
* or FILE1-DYN, FILE1-ERR section can be invoked.
* If there is an exception on opening FILE1-SEQ, INPUT-ERR
* section can be invoked. Since there is no USE procedure declared for the EXTEND mode or for FILE1-EXT, an exception on opening that file will cause an abnormal termination of the program. Also, since FILE1-SEQ in the fragment is not opened for OUTPUT mode, the OUTPUT-ERR USE procedure is not eligible to be invoked here.

********************************************

PROCEDURE DIVISION.
DECLARATIVES.
INPUT-ERR SECTION.
USE AFTER STANDARD ERROR PROCEDURE ON INPUT.
INP-1.
DISPLAY "INVOKED USE PROCEDURE FOR INPUT".
OUTPUT-ERR SECTION.
USE AFTER STANDARD ERROR PROCEDURE ON OUTPUT.
OUT-1.
DISPLAY "INVOKED USE PROCEDURE FOR OUTPUT".
FILE1-ERR SECTION.
USE AFTER STANDARD ERROR PROCEDURE ON FILE1-RAN, FILE1-DYN.
FILE1-1.
DISPLAY "INVOKED USE PROCEDURE FOR FILES".

END DECLARATIVES.
MAIN-PROGRAM SECTION.
P0. DISPLAY "***ENTERED USE TEST PROGRAM FRAGMENT***".

OPEN INPUT FILE1-SEQ.
OPEN OUTPUT FILE1-RAN.
OPEN I-O FILE1-DYN.
OPEN EXTEND FILE1-EXT.

...

WRITE

WRITE

Function

The WRITE statement releases a logical record to an output or input-output file. It can also position lines vertically on a logical page.

[rec-name]

is the name of a logical record described in the Data Division File Section. The logical record cannot be in a sort-merge file description entry.

[src-item]

is the identifier of the data item that contains the data.

[advance-num]
Procedure Division

is an integer or the identifier of an unsigned data item described as an integer. Its value can be zero.

[top-of-page-name]

is a mnemonic-name equated to C01 in the SPECIAL-NAMES paragraph of the Environment Division. It represents top-of-page and is equivalent to the PAGE phrase.

[stmt]

is an imperative statement executed when the relevant condition (end-of-page or invalid key) occurs.

[stmt2]

is an imperative statement executed when the relevant condition (not at end-of-page or not invalid key) occurs.

Syntax Rules

1. Format 1 must be used for sequential files.

2. Format 2 must be used for relative and indexed files.

3. If the file description entry containing rec-name has a LINAGE clause, the WRITE statement cannot have an ADVANCING top-of-page-name phrase.

4. If there is an END-OF-PAGE phrase, the file description entry containing rec-name must have a LINAGE clause.

5. The words END-OF-PAGE and EOP are equivalent.

6. In Format 2, there must be an INVALID KEY phrase if there is no applicable USE AFTER EXCEPTION procedure for the file.

7. To use the ALLOWING option, the program must include these entries:
   - LOCK-HOLDING clause of the I-O-CONTROL paragraph
   - ALLOWING option of the OPEN statement

8. If src-item is a function-identifier, it must reference an alphanumeric function. When src-item is not a function-identifier, rec-name and src-item must not reference the same storage area.

9. The ADVANCING PAGE and END-OF-PAGE phrase cannot be used in the same WRITE statement.

10. ADVANCING cannot be used with LINE SEQUENTIAL (Alpha, I64).

11. The ALLOWING clause is VSI standard file-sharing syntax, and cannot be used for a file connector that has had X/Open standard file-sharing syntax (WITH [NO] LOCK or LOCK MODE) specified.

General Rules

All Files

1. The record is no longer available in rec-name after a WRITE statement successfully executes. However, if the associated file-name is in a SAME RECORD AREA clause, the record is available in rec-name. In this case, the record is also available in the record areas of other file-names in the same SAME RECORD AREA clause.

2. The FROM Phrase section lists the rules for the FROM phrase.

3. For mass storage files, the WRITE statement does not affect the File Position Indicator.
4. The WRITE statement updates the value of the FILE STATUS data item for the file.

5. A file's maximum record size is set when it is created. It cannot be changed later.

6. On a mass storage device, the number of characters required to store a logical record in a file depends on file organization and record type. (See Technical Notes.)

7. WRITE statement execution releases a logical record to the I-O system.

8. The ALLOWING NO OTHERS option can be used only in a VSI standard manual record-locking environment. To create a manual record-locking environment, the program must open file-name with an ALLOWING option and specify the APPLY LOCK-HOLDING phrase of the I-O-CONTROL paragraph. If you use manual locking (APPLY LOCK-HOLDING), then the ALLOWING NO OTHERS clause on the WRITE statement is required.

9. The ALLOWING NO OTHERS option locks the current record. No other concurrent access stream can access this record until it is unlocked.

However, on Tru64 UNIX systems, for indexed files the WRITE statement with the ALLOWING clause does not acquire a record lock.

10. If there is an applicable USE AFTER EXCEPTION procedure, it executes whenever an input or output condition occurs that would result in a nonzero value in the first character of a FILE STATUS data item. However, it does not execute if: (a) the condition is invalid key, and (b) there is an INVALID KEY phrase.

See the rules for the INVALID KEY phrase, Section 6.6.10, “INVALID KEY Phrase”.

11. The number of character positions in the record to be written must not be less than the lowest or greater than the highest number of character positions allowed by the RECORD VARYING clause. In either case, the WRITE statement is unsuccessful and the following occurs:

• The WRITE operation does not take place.

• The contents of the record area remain unaffected.

• The I-O status of the file is set to a value that indicates the cause of the condition.

**Sequential or Line Sequential (Alpha, I64) Files**

12. The file must be open in the OUTPUT or EXTEND mode when the WRITE statement executes. (See Table 6.15, “Allowable Input-Output Statements for Sequential, Line Sequential (Alpha, I64), Relative, and Indexed Files.”)

13. The sequence of records in a sequential file is set by the order of WRITE statement executions that create the file. The relationship does not change, except when records are added to the end of the file.

14. For a sequential file open in the extend mode, the WRITE statement adds records to the end of the file as if the file were open in the output mode. If the file has records, the first record written after execution of an OPEN statement with the EXTEND phrase is the successor of the file's last record.

15. When a program tries to write beyond a sequential file's externally defined boundaries (for example, attempting to write to a full disk device), an exception condition exists as follows:

• The contents of the record area are unaffected.

• The value of the FILE STATUS data item for the file indicates a boundary violation.

• If a USE AFTER EXCEPTION procedure applies to the file, it executes.

• If there is no applicable USE AFTER EXCEPTION procedure, the program terminates abnormally.

16. If the end of a reel/unit is recognized, and the WRITE does not exceed the externally defined file boundaries:
• A reel/unit swap occurs.

• The Current Volume Pointer points to the file's next reel/unit.

17. If the program reaches the end of the logical page during execution of a WRITE statement with the END-OF-PAGE phrase, \textit{stment} executes. The LINAGE clause associated with the file specifies the logical end.

18. An end-of-page condition is reached when a WRITE statement with the END-OF-PAGE phrase causes printing or spacing in the footing area of the page body.

   This condition occurs when the WRITE statement causes the LINAGE-COUNTER to equal or exceed the value in the LINAGE clause FOOTING phrase. \textit{stment} then executes after \textit{rec-name} is written to the file.

   If this statement does not occur and the NOT END-OF-PAGE is specified, \textit{rec-name} is written to the file, file status is updated, and control is transferred to \textit{stment2}.

19. An automatic page overflow condition occurs when the page body cannot fully accommodate a WRITE statement (with or without the END-OF-PAGE phrase).

   This condition occurs when WRITE statement execution would cause the LINAGE-COUNTER to exceed the number of lines in the page body specified in the LINAGE clause. When this happens, the line is presented on the logical line before or after (depending on the phrase) device positioning. The device is positioned to the first line that can be written on the next logical page (as described in the LINAGE clause). \textit{stment} then executes after \textit{rec-name} is written to the file.

20. If there is no LINAGE clause FOOTING phrase, the WRITE statement operates as if the FOOTING phrase value was beyond the limits of the page. That is, the end-of-page condition occurs after the specified number of lines per page are written. No space is reserved for a footing.

21. If there is a FOOTING phrase, and a WRITE statement would cause the LINAGE-COUNTER to exceed both the number of lines in a logical page and the value in the LINAGE clause FOOTING phrase, the WRITE statement operates as if there were no FOOTING phrase.

**Relative Files**

22. The file must be open in the OUTPUT, I-O, or EXTEND mode when the WRITE statement executes. (See Table 6.15, “Allowable Input-Output Statements for Sequential, Line Sequential (Alpha, I64), Relative, and Indexed Files”.)

23. When a relative file with sequential access mode is open in the output mode, the WRITE statement releases a record to the I-O system. The first record has a relative record number of 1. Subsequent records have relative record numbers of 2, 3, 4, and so on. If \textit{rec-name} has an associated RELATIVE KEY data item, the WRITE places the relative record number of the released record into it.

24. When a relative file with sequential access mode is open in the extend mode, the WRITE statement releases a record to the I-O system. The first record has a relative record number one greater than the highest relative record number existing in the file. Subsequent records have consecutively higher relative record numbers. If \textit{rec-name} has an associated RELATIVE KEY data item, the WRITE statement places the relative record number of the released record into it.

25. When a relative file with random or dynamic access mode is open in the output mode, the program must place a value in the RELATIVE KEY data item before executing the WRITE statement. The value is the relative record number to associate with the record in \textit{rec-name}. The WRITE statement releases the record to the I-O system.

26. When a relative file is open in the I-O mode and the access mode is random or dynamic, the program must place a value in the RELATIVE KEY data item before executing the WRITE statement. The value is the relative record number to associate with the record in \textit{rec-name}. Executing a Format 2 WRITE statement releases the record to the I-O system.

27. The invalid key condition exists when:
• The access mode is random or dynamic, and the RELATIVE KEY data item specifies a record that already exists in the file.

• The program tries to write a record beyond the externally defined file boundaries.

• The number of significant digits in the relative record number is larger than the size of the relative key data item described for the file.

28. When the program detects an invalid key condition, WRITE statement execution is unsuccessful. The following results occur:

• The contents of the current record area are not affected.

• The WRITE statement sets the FILE STATUS data item for the file to indicate the cause of the condition.

• Program execution continues according to the rules for the invalid key condition.

See the rules for the INVALID KEY phrase, Section 6.6.10, “INVALID KEY Phrase”.

Indexed Files

29. The file must be open in the OUTPUT, I-O, or EXTEND mode when the WRITE statement executes. (See Table 6.15, “Allowable Input-Output Statements for Sequential, Line Sequential (Alpha, I64), Relative, and Indexed Files”.)

30. Executing a Format 2 WRITE statement releases a record to the I-O system. The contents of the record keys enable later record access based on any defined key.

31. When the file description entry has a RECORD KEY IS clause, the prime record key value is unique unless the DUPLICATES phrase is specified. When a program later accesses these records sequentially, the retrieval order is the same as the order in which they were written in the program.

32. The program must set the value of the prime record key data item before executing the WRITE statement.

33. If the file is open in the sequential access mode, the program must release records in ascending or descending order of prime record key values, depending on the sort order specified in the RECORD KEY clause. If the file is open in the extend mode, the first released record must have a prime record key value that logically follows the last record in the file according to the prime key sort order.

34. If the file is open in the random or dynamic access mode, the program can release records in any order.

35. When the file description entry has an ALTERNATE RECORD KEY clause, the alternate record key value is unique unless the program specifies the DUPLICATES phrase. When a program later accesses these records sequentially, the retrieval order is the same as the order in which they were written in the program.

36. The invalid key condition occurs for any of the following:

• The file is open in the sequential access mode and in the OUTPUT or EXTEND mode, and the prime record key value does not logically follow the prime record key value of the previous record.

• The file is open in the OUTPUT, EXTEND, or I-O mode, the prime record key value duplicates an existing record's prime record key value, and the program does not specify duplicates on the prime record key.

• The file is open in the OUTPUT, EXTEND, or I-O mode, and the value of an alternate record key (for which duplicates are not allowed) duplicates the value of the corresponding data item in an existing record.

• The program tries to write a record beyond the externally defined file boundaries.

37. When the program detects an invalid key condition, WRITE statement execution is unsuccessful. The following results occur:
• The contents of the current record area are not affected.

• The WRITE statement sets the FILE STATUS data item for the file to indicate the cause of the condition.

• Program execution continues according to the rules for the invalid key condition.

See the rules for the INVALID KEY phrase, Section 6.6.10, “INVALID KEY Phrase”.

**Technical Notes**

• WRITE statement execution can result in these FILE STATUS data item values:

<table>
<thead>
<tr>
<th>File Status</th>
<th>File Organization</th>
<th>Access Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>All</td>
<td>All</td>
<td>Write is successful</td>
</tr>
<tr>
<td>02</td>
<td>Ind</td>
<td>All</td>
<td>Created duplicate primary or alternate key</td>
</tr>
<tr>
<td>21</td>
<td>Ind</td>
<td>Seq</td>
<td>Attempted key value not in prime key sort order (invalid key)</td>
</tr>
<tr>
<td>22</td>
<td>Ind, Rel</td>
<td>All</td>
<td>Duplicate key (invalid key)</td>
</tr>
<tr>
<td>24</td>
<td>Ind, Rel</td>
<td>All</td>
<td>Boundary violation (relative or indexed files) or relative record number is too large for relative key data item (invalid key)</td>
</tr>
<tr>
<td>34</td>
<td>Seq</td>
<td>Seq</td>
<td>Boundary violation (sequential files)</td>
</tr>
<tr>
<td>44</td>
<td>All</td>
<td>All</td>
<td>Boundary violation. An attempt was made to write a record that is larger than the largest or smaller than the smallest record allowed</td>
</tr>
<tr>
<td>48</td>
<td>All</td>
<td>All</td>
<td>File not open, or incompatible open mode</td>
</tr>
<tr>
<td>92</td>
<td>Ind, Rel</td>
<td>All</td>
<td>Record locked by another process</td>
</tr>
<tr>
<td>30</td>
<td>All</td>
<td>All</td>
<td>All other permanent errors</td>
</tr>
</tbody>
</table>

In order to detect "device full" (file status 34) on a sequential WRITE operation, each WRITE needs to be followed by a call to SYSS$FLUSH to ensure that an attempt has been made to write any buffered records to disk. For more information, at the OpenVMS system prompt, type

```
HELP RMS $FLUSH
```

**Additional References**

• LOCK-HOLDING Phrase in Section : APPLY Clause in I-O-CONTROL paragraph in Chapter 4, Environment Division

• Section 6.1.4: Scope of Statements

• Section 6.6.8: I-O Status
• Section 6.6.10: INVALID KEY Phrase
• Section 6.6.11: FROM Phrase
• OPEN statement
• UNLOCK statement

END PROGRAM

Function
The END PROGRAM header indicates the end of the named COBOL source program. Alternatively, the end of a named COBOL source program can be indicated by the end of the program's Procedure Division.

[program-name]

must contain 1 to 31 characters and follow the rules for user-defined words. It must be identical to a program-name declared in a preceding PROGRAM-ID paragraph.

Syntax Rules
1. An inside PROGRAM-ID/END PROGRAM pair must be contained within the outside pair.
2. The END PROGRAM header must be present in every program that either contains or is contained within another program.
3. The END PROGRAM header indicates the end of a specific COBOL source program.
4. The END PROGRAM header starts in Area A.
5. The only COBOL statements that can follow an END PROGRAM header are as follows:
   • An Identification Division header of another program
   • Another END PROGRAM header
   The last END PROGRAM header must reference the outermost containing program.
6. If a program includes an END PROGRAM header and if it is not contained in another program, the next COBOL statement, if any, must be the Identification Division header of another program to be compiled.

Additional Reference
PROGRAM-ID paragraph in Chapter 3, Identification Division

Examples
1. This separately compiled program (PROG-NAME-A) contains one program (PROG-NAME-B).

   IDENTIFICATION DIVISION.
   PROGRAM-ID. PROG-NAME-A.  
   PROCEDURE DIVISION.  
   ...
2. This separately compiled program (PROG-NAME-A) contains eight other programs through. Also, is contained within , and is contained within , and are contained within , and are directly contained within .

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-A.

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-B.

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-C.

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-D.

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-F.

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-G.

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-H.

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-I.

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG-NAME-J.
Chapter 7. Intrinsic Functions

Data processing problems frequently require the use of values not directly accessible in the data storage associated with a program. These data values must be derived through operations on other data. Instead of having to write code to specify many common operations step by step, the programmer can use intrinsic functions. An intrinsic function is treated as a temporary elementary data item that contains a temporary data value to be derived automatically at the time of reference during execution of the program.

The uses of the intrinsic functions can be summarized briefly in the following listing by category:

<table>
<thead>
<tr>
<th>CATEGORY</th>
<th>FUNCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scientific/Mathematical</td>
<td>ACOS, ASIN, ATAN, COS, FACTORIAL, LOG, LOG10, MOD, REM, SIN, SQRT, SUM, TAN</td>
</tr>
<tr>
<td>Relational</td>
<td>MAX, MIN, ORD-MAX, ORD-MIN</td>
</tr>
<tr>
<td>String Manipulation</td>
<td>LOWER-CASE, NUMVAL, NUMVAL-C, REVERSE, UPPER-CASE</td>
</tr>
<tr>
<td>Date Manipulation</td>
<td>CURRENT-DATE, DATE-OF-INTEGER, DATE-TO-YYYYMMDD, DAY-TO-YYYYDDDD, DAY-OF-INTEGER, INTEGER-OF-DATE, INTEGER-OF-DAY, TEST-DATE-YYYYMMDD, TEST-DAY-YYYYDDDD, WHEN-COMPILED, YEAR-TO-YYYY</td>
</tr>
<tr>
<td>Statistical/Accounting</td>
<td>ANNUITY, MEAN, MEDIAN, MIDRANGE, PRESENT-VALUE, RANGE, STANDARD-DEVIATION, VARIANCE</td>
</tr>
<tr>
<td>Other</td>
<td>ARGCOUNT (OpenVMS), CHAR, INTEGER, INTEGER-PART, LENGTH, ORD, RANDOM</td>
</tr>
</tbody>
</table>

Later in this chapter (in : Function Descriptions) you will find a comprehensive table (Table 7.1, “Intrinsic Functions”) of functions, including their types, arguments, and values returned. Following the table are complete descriptions, including formats, of the individual functions in alphabetic order.

Intrinsic Function

Intrinsic Function

Description

A call to an intrinsic function is constructed as a function-identifier made up of the word FUNCTION and a name, as well as any applicable arguments and modifiers. The name is one of those shown in Table 7.1, “Intrinsic Functions”. An argument (see the description in the argument section) is selected according to application requirements.

A function-identifier is a syntactically correct combination of character strings and separators that uniquely references the data item resulting from the evaluation of a function. Although intrinsic functions are treated as elementary data items, they cannot be receiving operands.

A function-identifier that references an alphanumeric function can generally be specified wherever a sending identifier is permitted and wherever a reference to a function is not specifically prohibited by general-format rules. (For example, the rules for the CALL statement prohibit a function from being referenced in a CALL statement as an argument.) An integer or numeric function can be used anywhere an arithmetic expression (defined in Section 6.4, “Arithmetic Expressions”) can be used.

General Format
FUNCTION function-name [({argument}...)] [reference-modifier]

[function-name]

is one of the names listed in the first column of Table 7.1, “Intrinsic Functions”. A function-name must be specified as part of a function-identifier. Most function-names are not reserved words, and can be used in a program outside the context of a function.

[argument]

is an identifier, a literal, or an arithmetic expression. It complies with the specific rules governing the number, class, and category of arguments for the function. If it is an identifier, it can be subscripted, qualified, or reference-modified, and it can be a function-identifier. Functions may have between 0 and 250 arguments as specified by the definition of each function. The arguments in an argument list may be separated by a comma. Arguments are evaluated individually, from left to right.

Most intrinsic functions require one or more arguments. The programmer must specify arguments of the proper type and number and within the legal constraints for the function; otherwise, the result of the statement may be undefined.

[reference-modifier]

can be specified only for alphanumeric functions. It specifies the beginning character position to be selected in the resulting data item and optionally the length of the resulting data item. (For more information on reference modification, see Section 6.2.3, “Reference Modification”.)

Functions and Subscripting

An argument of an intrinsic function that permits a variable number of arguments can be a generically subscripted table or portion of a table. Generic, or ALL, subscripting (available only for intrinsic function arguments) is the use of the word ALL to specify all elements in one or more dimensions of a table. (A table element is a data item that contains or is subordinate to an OCCURS clause; if it is an OCCURS DEPENDING ON clause, the range of values is determined by the object of the clause.) Additional arguments, if any, of the function may or may not be table names. The evaluation of an ALL subscript must result in at least one argument; otherwise the result of executing the statement is undefined.

The order of the implicit specification of each occurrence of a table element is from left to right. This process is spelled out in detail in the following paragraph:

The first (or leftmost) specification is the identifier with each subscript specified by ALL replaced by one, and the next specification is the same identifier with the rightmost subscript specified by ALL incremented by one. This process continues with the rightmost ALL subscript being incremented by one for each implicit specification until the rightmost ALL subscript has been incremented through its range of values. If there are any additional ALL subscripts, the ALL subscript immediately to the left of the rightmost ALL is incremented by one, the rightmost ALL is reset to one, and the process of varying the rightmost ALL subscript is repeated. The ALL subscript to the left of the rightmost ALL subscript is incremented by one through its range of values. For each additional ALL subscript, this process is repeated in turn until the leftmost ALL subscript has been incremented by one through its range of values. If the ALL subscript is associated with an OCCURS DEPENDING ON clause, the range of values is determined by the object of that clause.

The reference modifier (if any) of an argument with an ALL subscript applies to each of the implicitly specified elements of the table. See Chapter 6: Procedure Division for the general format of ALL subscripting.

When one subscript of a multidimensional table is ALL, every other subscript must be one of the following:

Another ALL subscript
A positive integer literal
The data-name of a numeric integer elementary item (optionally followed by a plus or minus sign and an integer literal)
An index-name (optionally followed by a plus or minus sign and an integer literal)
Intrinsic Functions

The functions that permit generic subscripting of arguments are the following:

MAX
MEAN
MEDIAN
MIDRANGE
MIN
ORD-MAX
ORD-MIN
PRESENT-VALUE
RANGE
STANDARD-DEVIAITION
SUM
VARIANCE

See MAX and SUM for examples of generically subscripted arguments.

Function Descriptions

There are three types of functions, based on the type of their resultant values, as follows:

- Alphanumeric—A function whose resultant value is composed of a string of one or more characters from the computer's character set, and whose implicit usage is DISPLAY

- Integer—A function whose resultant value is a number that cannot have any nonzero digits to the right of the decimal point

- Numeric—A function whose resultant value is a number that may have nonzero digits to the right of the decimal point

Table 7.1, “Intrinsic Functions” lists the intrinsic functions, along with their types, their arguments, and the values they return. Complete descriptions of the functions, arranged alphabetically, follow.

Table 7.1. Intrinsic Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Number and Type of Arguments</th>
<th>Function Type</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOS</td>
<td>1 numeric, num</td>
<td>Numeric</td>
<td>Arcosine of num</td>
</tr>
<tr>
<td>ANNUITY</td>
<td>1 numeric, num, and 1 integer, int</td>
<td>Numeric</td>
<td>Ratio of annuity paid for each of int periods at interest of num to initial investment of one monetary unit</td>
</tr>
<tr>
<td>ARGCOUNT (OpenVMS)</td>
<td>None</td>
<td>Integer</td>
<td>Number of arguments passed to the COBOL program</td>
</tr>
<tr>
<td>ASIN</td>
<td>1 numeric, num</td>
<td>Numeric</td>
<td>Arcsine of num</td>
</tr>
<tr>
<td>ATAN</td>
<td>1 numeric, num</td>
<td>Numeric</td>
<td>Arctangent of num</td>
</tr>
<tr>
<td>CHAR</td>
<td>1 integer, int</td>
<td>Alphanumeric</td>
<td>Character in position int of program collating sequence</td>
</tr>
<tr>
<td>COS</td>
<td>1 numeric, num</td>
<td>Numeric</td>
<td>Cosine of num</td>
</tr>
<tr>
<td>CURRENT-DATE</td>
<td>None</td>
<td>Alphanumeric</td>
<td>Current date and time</td>
</tr>
<tr>
<td>DATE-OF-INTEGER</td>
<td>1 integer</td>
<td>Integer</td>
<td>Standard date equivalent (YYYYMMDD) of integer date 5</td>
</tr>
<tr>
<td>Function</td>
<td>Number and Type of Arguments</td>
<td>Function Type</td>
<td>Value Returned</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------</td>
<td>---------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DATE-TO-YYYYMMDD</td>
<td>1 or 2 integer</td>
<td>Integer</td>
<td>YYYYMMDD date converted from YYMMD date</td>
</tr>
<tr>
<td>DAY-OF-INTEGER</td>
<td>1 integer</td>
<td>Integer</td>
<td>YYYYYDDD date equivalent of integer date</td>
</tr>
<tr>
<td>DAY-TO-YYYYDDD</td>
<td>1 or 2 integer</td>
<td>Integer</td>
<td>YYYYYDDD date converted from YYDDD date</td>
</tr>
<tr>
<td>FACTORIAL</td>
<td>1 integer, int</td>
<td>Integer</td>
<td>Factorial of int</td>
</tr>
<tr>
<td>INTEGER</td>
<td>1 numeric, num</td>
<td>Integer</td>
<td>The greatest integer not greater than num</td>
</tr>
<tr>
<td>INTEGER-OF-DAY</td>
<td>1 integer</td>
<td>Integer</td>
<td>Integer date equivalent of standard date (YYYYMMDD)</td>
</tr>
<tr>
<td>INTEGER-OF-DAY</td>
<td>1 integer</td>
<td>Integer</td>
<td>Integer date equivalent of date in YYYYDDD format</td>
</tr>
<tr>
<td>INTEGER-PART</td>
<td>1 numeric, num</td>
<td>Integer</td>
<td>Integer part of num</td>
</tr>
<tr>
<td>LENGTH</td>
<td>1 alphabetic or numeric or alphanumeric data item, or 1 nonnumeric literal</td>
<td>Integer</td>
<td>Length of argument</td>
</tr>
<tr>
<td>LOG</td>
<td>1 numeric, num</td>
<td>Numeric</td>
<td>Natural logarithm of num</td>
</tr>
<tr>
<td>LOG10</td>
<td>1 numeric, num</td>
<td>Numeric</td>
<td>Logarithm to base 10 of num</td>
</tr>
<tr>
<td>LOWER-CASE</td>
<td>1 alphabetic or 1 alphanumeric</td>
<td>Alphanumeric</td>
<td>All letters in the argument set to lowercase</td>
</tr>
<tr>
<td>MAX</td>
<td>1 or more alphabetic and/or alphanumeric, or 1 or more integer and/or numeric</td>
<td>Depends on arguments</td>
<td>Value of maximum argument</td>
</tr>
<tr>
<td>MEAN</td>
<td>1 or more numeric</td>
<td>Numeric</td>
<td>Arithmetic mean of arguments</td>
</tr>
<tr>
<td>MEDIAN</td>
<td>1 or more numeric</td>
<td>Numeric</td>
<td>Median of arguments</td>
</tr>
<tr>
<td>MIDRANGE</td>
<td>1 or more numeric</td>
<td>Numeric</td>
<td>Mean of minimum and maximum arguments</td>
</tr>
<tr>
<td>MIN</td>
<td>1 or more alphabetic and/or alphanumeric, or 1 or more integer and/or numeric</td>
<td>Depends on arguments</td>
<td>Value of minimum argument</td>
</tr>
<tr>
<td>MOD</td>
<td>2 integer, int-1 and int-2</td>
<td>Integer</td>
<td>Value of int-1 modulo int-2</td>
</tr>
<tr>
<td>NUMVAL</td>
<td>1 alphanumeric</td>
<td>Numeric</td>
<td>Numeric value of simple numeric string</td>
</tr>
<tr>
<td>NUMVAL-C</td>
<td>1 or 2 alphanumeric</td>
<td>Numeric</td>
<td>Numeric value of numeric string with optional commas and currency sign</td>
</tr>
<tr>
<td>Function</td>
<td>Number and Type of Arguments</td>
<td>Function Type</td>
<td>Value Returned</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ORD</td>
<td>1 alphabetic or 1 alphanumeric</td>
<td>Integer</td>
<td>Ordinal position of the argument in collating sequence</td>
</tr>
<tr>
<td>ORD-MAX</td>
<td>1 or more alphabetic, or 1 or more numeric, or 1 or more alphanumeric</td>
<td>Integer</td>
<td>Ordinal position of maximum argument</td>
</tr>
<tr>
<td>ORD-MIN</td>
<td>1 or more alphabetic, or 1 or more numeric, or 1 or more alphanumeric</td>
<td>Integer</td>
<td>Ordinal position of minimum argument</td>
</tr>
<tr>
<td>PRESENT-VALUE</td>
<td>1 numeric, num-1; and 1 or more additional numeric, num-2</td>
<td>Numeric</td>
<td>Present value of a series of future period-end amounts, num-2, at a discount rate of num-1</td>
</tr>
<tr>
<td>RANDOM</td>
<td>1 integer or none</td>
<td>Numeric</td>
<td>Pseudo-random number</td>
</tr>
<tr>
<td>RANGE</td>
<td>1 or more integer, or 1 or more numeric</td>
<td>Integer or numeric depending on arguments</td>
<td>Value of maximum argument minus value of minimum argument</td>
</tr>
<tr>
<td>REM</td>
<td>2 numeric, num-1 and num-2</td>
<td>Numeric</td>
<td>Remainder of num-1/ num-2</td>
</tr>
<tr>
<td>REVERSE</td>
<td>1 alphabetic or 1 alphanumeric</td>
<td>Alphanumeric</td>
<td>Reverse order of the characters of the argument</td>
</tr>
<tr>
<td>SIN</td>
<td>1 numeric, num</td>
<td>Numeric</td>
<td>Sine of num</td>
</tr>
<tr>
<td>SQRT</td>
<td>1 numeric, num</td>
<td>Numeric</td>
<td>Square root of num</td>
</tr>
<tr>
<td>STANDARD-DEVIATION</td>
<td>1 or more numeric</td>
<td>Numeric</td>
<td>Standard deviation of arguments</td>
</tr>
<tr>
<td>SUM</td>
<td>1 or more integer or 1 or more numeric</td>
<td>Integer or numeric depending on arguments</td>
<td>Sum of arguments</td>
</tr>
<tr>
<td>TAN</td>
<td>1 numeric, num</td>
<td>Numeric</td>
<td>Tangent of num</td>
</tr>
<tr>
<td>TEST-DATE-YYYYMMDD</td>
<td>1 integer</td>
<td>Integer</td>
<td>0,1,2, or 3, indicating whether the date is a valid date in the Gregorian calendar, and reason if invalid</td>
</tr>
<tr>
<td>TEST-DAY-YYYYDDD</td>
<td>1 integer</td>
<td>Integer</td>
<td>0, 1, or 2, indicating whether the Julian date is a valid date in the Gregorian calendar, and reason if invalid</td>
</tr>
<tr>
<td>UPPER-CASE</td>
<td>1 alphabetic or 1 alphanumeric</td>
<td>Alphanumeric</td>
<td>All letters in the argument set to uppercase</td>
</tr>
<tr>
<td>VARIANCE</td>
<td>1 or more numeric</td>
<td>Numeric</td>
<td>Variance of argument</td>
</tr>
<tr>
<td>WHEN-Compiled</td>
<td>None</td>
<td>Alphanumeric</td>
<td>Date and time program was compiled</td>
</tr>
<tr>
<td>YEAR-TO-YYYY</td>
<td>1 or 2 integer</td>
<td>Integer</td>
<td>4-digit year, converted from 2-digit year</td>
</tr>
</tbody>
</table>

5An integer date is a positive integer representing the number of days after December 31, 1600, in the Gregorian calendar.

6A function that has only alphabetic and/or alphanumeric arguments is type alphanumeric. A function that has only integer arguments is type integer. A function that has both integer and numeric arguments is type numeric.
ACOS

Description
The ACOS function returns a numeric value in radians that approximates the arccosine of the argument.

General Format
FUNCTION ACOS (arg)
[arg]

Rules
1. The type of this function is numeric.
2. The returned value is the approximation of the arccosine of arg and is greater than or equal to 0 and less than or equal to π (approximately 3.14159).

Example
COMPUTE RSULT = FUNCTION ACOS (.85).
The value returned and stored in RSULT (a numeric data item) is a number that approximates the arccosine of .85.

ANNUITY

Description
The ANNUITY function (annuity immediate) returns a numeric value that approximates the ratio of an annuity paid at the end of each period for the number of periods specified (by the second argument) to an initial investment of one. Interest is earned at the rate specified (by the first argument), and is applied at the end of the period, before the payment.

General Format
FUNCTION ANNUITY (interest-rate num-periods)

Rules
1. The type of this function is numeric.
2. When the value of interest-rate is 0, the value of the function is the approximation of $1 / \text{num-periods}$.

3. When the value of interest-rate is not 0, the value of the function is the approximation of $\text{interest-rate} / (1 - (1 + \text{interest-rate})^{-\text{num-periods}})$.

**Example**

```cobol
COMPUTE RESULT = FUNCTION ANNUITY (INTEREST-RATE, NUM-PERIODS).
```

INTEREST-RATE is a numeric data item, and NUM-PERIODS is a numeric integer data item. If the value of INTEREST-RATE is 0 and the value of NUM-PERIODS is 6, RESULT has a value approximating $1/6$. If the value of INTEREST-RATE is .11 and the value of NUM-PERIODS is 6, RESULT (a numeric data item) has a value of approximately 0.2364.

**ARGCOUNT (OpenVMS Only)**

**Description**

The ARGCOUNT function returns a numeric integer equal to the number of arguments passed to the VSI COBOL program.

**General Format**

```cobol
FUNCTION ARGCOUNT
```

**Rules**

1. The type of this function is integer.

2. The returned value represents the actual number of arguments passed to the VSI COBOL program that contains the function.

**Example**

```cobol
IF FUNCTION ARGCOUNT = 3
  PERFORM PROCESS-OPTIONAL-3RD-ARGUMENT.
```

If there are three arguments passed to the VSI COBOL program containing the ARGCOUNT function, a third argument supplied with the COBOL program calling command will be parsed and processed.

**Additional Reference**

Refer to the Argument Information Register in the OpenVMS Calling Standard.

**ASIN**

**Description**

The ASIN function returns a numeric value in radians that approximates the arcsine of the argument.
General Format

FUNCTION ASIN (arg)

[arg]

is a numeric argument with a value greater than or equal to -1 and less than or equal to +1.

Rules

1. The type of this function is numeric.

2. The returned value is the approximation of the arcsine of arg and is greater than or equal to $-\pi/2$ and less than or equal to $+\pi/2$. ($\pi$ is approximately 3.14159.)

Example

COMPUTE RSULT = FUNCTION ASIN (.675).

The value returned and stored in RSULT (a numeric data item) is a number that approximates the arcsine of .675.

ATAN

ATAN

Description

The ATAN function returns a numeric value in radians that approximates the arctangent of the argument.

General Format

FUNCTION ATAN (arg)

[arg]

is a numeric argument.

Rules

1. The type of this function is numeric.

2. The returned value is the approximation of the arctangent of arg and is greater than $-\pi/2$ and less than $+\pi/2$. ($\pi$ is approximately 3.14159.)

Example

COMPUTE RSULT = FUNCTION ATAN (MEASUREMENT-IN-RADIANS).

MEASUREMENT-IN-RADIANS and RSULT are numeric data items. The value returned and stored in RSULT is a number that approximates the arctangent of the value of MEASUREMENT-IN-RADIANS.

CHAR

CHAR
Intrinsic Functions

Description

The CHAR function returns a one-character alphanumeric value that is a character in the program collating sequence having the ordinal position equal to the value of the argument.

General Format

FUNCTION CHAR (position)

[position]

is a positive integer argument representing the ordinal position of the desired character in the program collating sequence, and having a value less than or equal to the number of positions in the collating sequence.

Rules

1. The type of this function is alphanumeric.

2. The character returned as the function value is the character in the program collating sequence. (See the information on the ALPHABET clause in Chapter 4, Environment Division.)

3. If more than one character has the same position in the program collating sequence, the character returned as the function value is that of the first literal specified for that character position in the ALPHABET clause.

Example

MOVE FUNCTION CHAR (13) TO FORM-FEED-CH.

The character occupying the 13th position in the program collating sequence (that is, a form feed in the default collating sequence) is returned and stored in FORM-FEED-CH, which is an alphanumeric data item one character in length. (The numeric representation of a character is not the same as its ordinal position. Numeric representation starts at 0, whereas ordinals start at 1. Always add 1 to the numeric value of the desired character.)

COS

Description

The COS function returns a numeric value that approximates the cosine of an angle or arc, expressed in radians, that is specified by the argument.

General Format

FUNCTION COS (angle)

[angle]

is a numeric argument having the value of the measurement in radians of an angle or arc.

Rules

1. The type of this function is numeric.

2. The returned value is the approximation of the cosine of angle, and is greater than or equal to -1 and less than or equal to +1.
Example

COMPUTE COSIN-RSLT = FUNCTION COS (X).

X and COSIN-RSLT are numeric data items. If the value of X is 3, the approximate cosine of an angle of 3 radians is moved to COSIN-RSLT.

CURRENT-DATE

Description

The CURRENT-DATE function returns a 21-character alphanumeric value that represents the calendar date and the time of day.

General Format

FUNCTION CURRENT-DATE

Rules

1. The type of this function is alphanumeric.

2. The contents of the character positions returned, numbered from left to right, are as follows:

<table>
<thead>
<tr>
<th>Character Positions</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-4</td>
<td>Four numeric digits of the year in the Gregorian calendar.</td>
</tr>
<tr>
<td>5-6</td>
<td>Two numeric digits of the month of the year, in the range 01 through 12.</td>
</tr>
<tr>
<td>7-8</td>
<td>Two numeric digits of the day of the month, in the range 01 through 31.</td>
</tr>
<tr>
<td>9-10</td>
<td>Two numeric digits of the hours past midnight, in the range 00 through 23.</td>
</tr>
<tr>
<td>11-12</td>
<td>Two numeric digits of the minutes past the hour, in the range 00 through 59.</td>
</tr>
<tr>
<td>13-14</td>
<td>Two numeric digits of the seconds past the minute, in the range 00 through 59.</td>
</tr>
<tr>
<td>15-16</td>
<td>Two numeric digits of the hundredths of a second past the second, in the range 00 through 99.</td>
</tr>
<tr>
<td>17-21</td>
<td>The value 00000. (Reserved for future use.)</td>
</tr>
</tbody>
</table>

Example

The COBOL syntax for this function (similar to the example) is common to all platforms:

MOVE FUNCTION CURRENT-DATE TO RSULT.

199701101652313200000

This is a sample value returned by the example CURRENT-DATE function. Reading from left to right, it shows

- The year, 1997
- The month, January
Intrinsic Functions

- The day of the month, the 10th
- The time of day, 16:52 (4:52 P.M.)
- The seconds, 31, and the hundredths of seconds, 32, after 16:52:31

**DATE-OF-INTEGER**

**Description**

The DATE-OF-INTEGER function converts a date from an integer date form representing the number of days after December 31, 1600, to standard date form (YYYYMMD).  

**General Format**

FUNCTION DATE-OF-INTEGER (num-days)  

[num-days]  

is a positive integer argument that represents a number of days succeeding December 31, 1600, in the Gregorian calendar.

**Rules**

1. The type of this function is integer.
2. The returned value represents the ISO Standard date, in the form YYYYMMDD, that is equivalent to the integer specified. YYYY is an integer in the range 1601 through 9999. MM is an integer in the range 1 through 12. DD is an integer in the range 1 through 31.

**Example**

COMPUTE RESULT = FUNCTION DATE-OF-INTEGER (20).

The value returned and stored in RESULT (a numeric integer data item) is  

16010120

This value represents January 20, 1601, which is 20 days after December 31, 1600.

**DATE-TO-YYYYMMDD**

**Description**

The DATE-TO-YYYYMMDD function converts a date in the form YYYMMDD to the form YYYYMMDD. An optional second argument, when added to the current year (at the time the program executes), defines the ending year of a 100-year interval. This interval determines to what century the two-digit year belongs.

**General Format**

FUNCTION DATE-TO-YYYYMMDD( arg-1 [ arg-2 ] )
Intrinsic Functions

[arg-1] is a nonnegative integer between 0 and 999999.

[arg-2] is an integer. Its value, when added to the current year, must be between 1700 and 9999. If it is omitted, the default value is 50.

Rules

1. The type of this function is integer.
2. The returned value is an integer representing YYYYMMDD and is calculated as follows:

   \[
   \begin{align*}
   YY & = \text{int}(\text{arg-1} / 10000) \\
   mmd & = \text{mod}(\text{arg-1}, 10000) \\
   \text{return FUNCTION YEAR-TO-YYYY}(YY, \text{arg-2}) \times 10000 + mmd
   \end{align*}
   \]

Example

IF FUNCTION DATE-TO-YYYYMMDD (801123, 50 ) = 19801123 
DISPLAY "correct".
IF FUNCTION DATE-TO-YYYYMMDD (801123, 100 ) = 20801123 
DISPLAY "correct".
IF FUNCTION DATE-TO-YYYYMMDD (801123, -100 ) = 18801123 
DISPLAY "correct".

DATE-TO-YYYYMMDD implements a sliding window algorithm. To use it for a fixed window, you can specify arg-2 as follows:

(fixed-ending-year - function numval (function current-date (1:4)))

If fixed-ending-year is 2100, then in 1999 arg-2 has the value 101. If arg-1 is 501123, the returned-value is 20501123. If arg-1 is 991123, the returned-value is 20991123.

DAY-OF-INTEGER

Description

The DAY-OF-INTEGER function converts a date from an integer date form representing the number of days succeeding December 31, 1600, to a date form (sometimes called "Julian") representing year and days (YYYYDDD).

General Format

FUNCTION DAY-OF-INTEGER (num-days)

[num-days] is a positive integer argument that represents a number of days succeeding December 31, 1600, in the Gregorian calendar.

Rules

1. The type of this function is integer.
2. The returned value is an integer of the form YYYYDDD, where YYYY represents a year in the Gregorian calendar and DDD represents the day of that year. YYYY is an integer in the range 1601 through 9999. DDD is an integer in the range 1 through 366.

**Example**

COMPUTE RESULT = FUNCTION DAY-OF-INTEGER (28).

The value returned and stored in RESULT (a numeric integer data item) shows the 28th day of the year 1601 (that is, 28 days succeeding December 31, 1600), as follows:

1601028

**DAY-TO-YYYYDDD**

**Description**

The DAY-TO-YYYYDDD function converts a date in the form YYDDD to the form YYYYDDD. An optional second argument, when added to the current year (at the time the program executes), defines the ending year of a 100-year interval. This interval determines to what century the two-digit year belongs.

**General Format**

FUNCTION DAY-TO-YYYYDDD ( arg-1 [ arg-2 ] )

[arg-1] is a nonnegative integer between 0 and 99999.

[arg-2] is an integer. Its value, when added to the current year, must be between 1700 and 9999. If it is omitted, the default value is 50.

**Rules**

1. The type of this function is integer.

2. The returned value is an integer representing YYYYDDD and is calculated as follows:

\[
\begin{align*}
YY &= \text{int}(\frac{\text{arg-1}}{1000}) \\
\text{ddd} &= \text{mod}(\text{arg-1}, 1000) \\
\text{return }& \text{FUNCTION YEAR-TO-YYYY}(YY, \text{arg-2}) \times 1000 + \text{ddd}
\end{align*}
\]

**Example**

IF FUNCTION DAY-TO-YYYYDDD (80111, 50 ) = 1980111
DISPLAY "correct".

IF FUNCTION DAY-TO-YYYYDDD (80111, 100 ) = 2080111
DISPLAY "correct".

IF FUNCTION DAY-TO-YYYYDDD (80111, -100 ) = 1880111
DISPLAY "correct".

DAY-TO-YYYYDDD implements a sliding window algorithm. To use it for a fixed window, you can specify arg-2 as follows:
(fixed-ending-year - function numval (function current-date (1:4)))

If fixed-ending-year is 2100, then for 1999 \( \text{arg-2} \) has the value 101. If \( \text{arg-1} \) is 50111, the returned-value is 2050111. If \( \text{arg-1} \) is 99111, the returned-value is 2099111.

**FACTORIAL**

**Description**

The FACTORIAL function returns an integer that is the factorial of the argument specified.

**General Format**

FUNCTION FACTORIAL (num)

[num]

is 0 or a positive integer argument whose value is less than or equal to 19.

**Rules**

1. The type of this function is integer.
2. If the value of the argument is 0, the value 1 is returned.
3. If the value of the argument is positive, its factorial is returned.

**Example**

```
COMPUTE RESULT = FUNCTION FACTORIAL (NUM).
```

NUM and RESULT are numeric integer data items. If NUM has the value of 5, the value returned and stored in RESULT is 120.

\( 5! = 5 \times 4 \times 3 \times 2 \times 1 = 120. \)

**INTEGER**

**Description**

The INTEGER function returns the greatest integer value that is less than or equal to the argument.

**General Format**

FUNCTION INTEGER (num)

[num]

is a numeric argument.
Rule

The type of this function is integer.

Example

COMPUTE RSULT = FUNCTION INTEGER (NUM).

If the value of NUM (a numeric data item) is -1.5, the value returned and stored in RSULT (a numeric integer data item) is -2, because -2 is the greatest integer that is less than or equal to -1.5. If the value of NUM is +1.5, the value returned and stored in RSULT is +1, because +1 is the greatest integer that is less than or equal to +1.5.

INTEGER-OF-DATE

Description

The INTEGER-OF-DATE function converts a date from standard date form (YYYYMMDD) to an integer date form representing the number of days after December 31, 1600.

General Format

FUNCTION INTEGER-OF-DATE (num)

[num]

is an integer argument of the form YYYYMMDD representing a date subsequent to December 31, 1600.

Rules

1. The type of this function is integer.

2. The value of the argument is obtained from the calculation (YYYY * 10,000) + (MM * 100) + DD. YYYY represents the year in the Gregorian calendar, and must be an integer in the range 1601 through 9999. MM represents a month and is an integer in the range 1 through 12. DD represents a day and is an integer in the range 1 through 31; the value of DD must be valid for the specified month and year combination.

3. The returned value is an integer that is the number of days the specified date succeeds December 31, 1600, in the Gregorian calendar.

Examples

1. COMPUTE RSULT = FUNCTION INTEGER-OF-DATE (NUM).

NUM and RSULT are numeric integer data items. If NUM has the value 16010215 (that is, February 15, 1601), the value returned and stored in RSULT is 46 (the 46th day after December 31, 1600).

2. COMPUTE DAYS-IN-BILLING-CYCLE =
   FUNCTION INTEGER-OF-DATE (THIS-ENDING-DATE) -
   FUNCTION INTEGER-OF-DATE (LAST-ENDING-DATE)

DAYS-IN-BILLING-CYCLE, THIS-ENDING-DATE, and LAST-ENDING-DATE are numeric integer items. If THIS-ENDING-DATE has the value 19970301 (representing March 1, 1997), and LAST-ENDING-DATE has the value 19970201 (representing February 1, 1997), the value returned is 28.
INTEGRER-OF-DAY

Description

The INTEGRER-OF-DAY function converts a date in the Gregorian calendar from year-day (YYYYDDD) form (sometimes called "Julian") to an integer date form representing the number of days after December 31, 1600.

General Format

FUNCTION INTEGER-OF-DAY (num)

[num]

is an integer argument of the form YYYYDDD representing a date subsequent to December 31, 1600.

Rules

1. The type of this function is integer.

2. The value of the argument is obtained from the calculation (YYYY * 1000) + DDD. YYYY represents the year in the Gregorian calendar, and must be an integer in the range 1601 through 9999. DDD represents a day and is an integer in the range 1 through 366; the value of DDD must be valid for the specified year.

3. The returned value is an integer that is the number of days the specified date succeeds December 31, 1600, in the Gregorian calendar.

Example

COMPUTE RESULT = FUNCTION INTEGER-OF-DAY (1601365).

The value returned and stored in RESULT (a numeric integer data item) is 365, which is the number of days succeeding December 31, 1600, and which represents December 31, 1601.

INTEGRER-PART

Description

The INTEGRER-PART function returns an integer that is the integer portion of the argument.

General Format

FUNCTION INTEGER-PART (num)

[num]

is a numeric argument.

Rules

1. The type of this function is integer.

2. If the value of the argument is 0, the returned value is 0.
Intrinsic Functions

3. If the value of the argument is positive, the returned value is the greatest integer less than or equal to the value of the argument.

4. If the value of the argument is negative, the returned value is the least integer greater than or equal to the value of the argument.

Example

COMPUTE RESULT = FUNCTION INTEGER-PART (NUM).

NUM is a numeric data item, and RESULT is a numeric integer data item. If NUM has the value 0, the value returned is 0. If NUM has the value +1.5, the value returned is +1. If NUM has the value -1.5, the value returned is -1 (the least integer greater than or equal to the value of -1.5).

LENGTH

Description

The LENGTH function returns an integer equal to the length of the argument in character positions.

FUNCTION LENGTH (arg)

[arg]

is a nonnumeric literal or a data item of any class or category.

Rules

1. The type of this function is integer.

2. The value returned is an integer equal to the length of the argument in character positions. However, if the argument is a group data item containing a variable occurrence data item, the returned value is an integer determined by evaluation of the data item specified in the DEPENDING phrase of the OCCURS clause for that variable occurrence data item. This evaluation is accomplished according to the rules in the OCCURS clause dealing with the data item as a sending data item.

3. The returned value includes implicit FILLER characters, if any.

4. For items that are not USAGE DISPLAY, the returned value represents the allocated physical storage in bytes as described in Tables 5.12 and 5.13.

Examples

1. COMPUTE RESULT = FUNCTION LENGTH ("J. R. Donaldson").

   The value 15 is returned and stored in RESULT (a numeric integer data item).

2. 01 RECORD-SIZE PIC 9(9).
   01 RECORD1.
      05 REC-TYPE PIC 9(4) VALUE 23.
      05 REC-CNT PIC 9(4) VALUE 50.
      05 A-REC PIC X(30) OCCURS 1 TO 100 TIMES DEPENDING ON REC-CNT.
Intrinsic Functions

**LOG**

**Description**

The LOG function returns a numeric value that approximates the logarithm to the base e (natural log) of the argument.

**General Format**

FUNCTION LOG (num)

[num]

is a positive numeric argument.

**Rules**

1. The type of this function is numeric.
2. The returned value is an approximation of the logarithm to the base e of the argument.

**Example**

COMPUTE RESULT = FUNCTION LOG (NUM).

NUM and RESULT are numeric data items; the value of NUM must be greater than 0. The value returned and stored in RESULT is an approximation of the logarithm to the base e of NUM.

**LOG10**

**Description**

The LOG10 function returns a numeric value that approximates the logarithm to the base 10 of the argument.

**General Format**

FUNCTION LOG10 (num)

[num]

is a positive numeric argument.

**Rules**

1. The type of this function is numeric.
2. The returned value is an approximation of the logarithm to the base 10 of the argument.

**Example**

```plaintext
COMPUTE RSULT = FUNCTION LOG10 (NUM).
```

NUM and RSULT are numeric data items; the value of NUM must be greater than 0. The value returned and stored in RSULT is an approximation of the logarithm to the base 10 of NUM.

**LOWER-CASE**

**Description**

The LOWER-CASE function returns a character string that is the same length as the argument with each uppercase letter in the argument replaced by the corresponding lowercase letter.

**General Format**

```plaintext
FUNCTION LOWER-CASE (string)

[string]

is an alphabetic or alphanumeric argument at least one character in length.
```

**Rules**

1. The type of this function is alphanumeric.
2. The returned value is the same character string as the argument, except that each uppercase letter in the argument is replaced by the corresponding lowercase letter.

**Example**

```plaintext
MOVE FUNCTION LOWER-CASE (STR) TO LC-STR.
```

If STR (an alphanumeric data item six characters in length) contains the value "Autumn" the value returned and stored in LC-STR (also an alphanumeric data item six characters in length) is "autumn"; if STR contains "fall98" the value returned is unchanged ("fall98").

**MAX**

**Description**

The MAX function returns the contents of the argument that contains the maximum value.

**General Format**

```plaintext
FUNCTION MAX (argument…)

[argument]
```
Intrinsic Functions

is an alphabetic, alphanumeric, integer, or numeric argument.

Rules

1. The arguments must be all alphabetic, all alphanumeric, all integer, or all numeric, except that integer and numeric arguments can be mixed and alphabetic and alphanumeric arguments can be mixed.

2. The type of the function depends on the arguments, as follows:

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Function Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabetic and/or alphanumeric</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>Integer (all arguments)</td>
<td>Integer</td>
</tr>
<tr>
<td>Numeric (some arguments might be integer)</td>
<td>Numeric</td>
</tr>
</tbody>
</table>

3. The returned value consists of the contents of the argument having the greatest value, as determined by comparisons made according to the rules for simple conditions. (See Chapter 6, Procedure Division.)

4. If more than one argument has the same value, and that value is the maximum, the returned value consists of the contents of the leftmost of these arguments.

5. If there is only one argument, the returned value consists of the contents of that argument.

6. If the type of the function is alphanumeric, the size of the returned value is the same as the size of the argument selected as the maximum.

Examples

1. MOVE FUNCTION MAX ("A", "B", "C") TO MAX-LETTER-OUT.
   MOVE FUNCTION MAX (1, 2, 3) TO MAX-NUMBER-OUT.

   MAX-LETTER-OUT is alphabetic or alphanumeric, and receives the value "C"; MAX-NUMBER-OUT is integer and receives the value 3.

2. COMPUTE ITEMC = (ITEMA + FUNCTION MAX (ITEMB, 10)).

   If ITEM A and ITEMB both contain the value 1, this statement results in ITEMC having the value 11.

   IF FUNCTION MAX (A, B, C) > 100 …

   This is equivalent to the following more complex code:

   IF A >= B
      IF A >= C
         MOVE A TO TMP
      ELSE
         MOVE C TO TMP
   ELSE
      IF B >= C
         MOVE B TO TMP
      ELSE
         MOVE C TO TMP.
   IF TMP > 100 …

3. The following example shows generic subscripting with reference modification:

   05 TABLE1 PIC X(7) OCCURS 3 TIMES.
   .
   .
**Intrinsic Functions**

MOVE "XAAAAAQ" TO TABLE1(1).
MOVE "XBBBBBQ" TO TABLE1(2).
MOVE "XCCCCCQ" TO TABLE1(3).
MOVE FUNCTION MAX(TABLE1(ALL)(2:5)) TO RESULT.

The value "CCCCC" is returned and stored in RESULT, an alphanumeric data item. The reference modifier, (2:5), applies to each element implicitly specified by the ALL subscript. Thus,

\[
\text{FUNCTION MAX(TABLE1(ALL)(2:5))}
\]

is equivalent to

\[
\text{FUNCTION MAX(TABLE1(1)(2:5), TABLE1(2)(2:5), TABLE1(3)(2:5))}
\]

**MEAN**

**Description**

The MEAN function returns a numeric value that is the arithmetic mean (average) of its arguments.

**General Format**

FUNCTION MEAN ({arg} ...)

[arg]

is a numeric argument.

**Rules**

1. The type of this function is numeric.

2. The return value is the arithmetic mean of the arguments in the argument list; that is, it is the sum of the arguments divided by the number of arguments.

**Examples**

1. COMPUTE AVERAGE-VALUE = FUNCTION MEAN (9, 2, 6, 7, 1).

   The value returned and stored in AVERAGE-VALUE (a numeric data item) is 5 (the sum of the arguments divided by the number of arguments).

2. COMPUTE MEAN-ANSWER = FUNCTION MEAN(A, B, C).

   MEAN-ANSWER, A, B, and C are numeric data items. This code is equivalent to

   COMPUTE MEAN-ANSWER = (A + B + C) / 3.

**MEDIAN**

MEDIAN
Intrinsic Functions

Description
The MEDIAN function returns the median value of a list of numbers, represented by the arguments. This value is such that at least half of the values are greater than or equal to the returned value, and at least half are less than or equal.

General Format
FUNCTION MEDIAN ({num}...)
[num]
is a numeric argument.

Rules
1. The type of this function is numeric.
2. If the number of arguments is odd, the returned value is the middle occurrence in the sorted list.
3. If the number of arguments is even, the returned value is the arithmetic mean of the values referenced by the two middle occurrences in the sorted list.
4. The comparisons used to arrange the argument values in sorted order are made according to the rules for simple conditions. (See Chapter 6, Procedure Division.)

Examples
1. COMPUTE RSULT = FUNCTION MEDIAN (1, 1, 9, 2, 1).
   The value returned and stored in RSULT (a numeric data item) is 1.
2. COMPUTE RSULT = FUNCTION MEDIAN (1, 1, 9, 2).
   The value returned and stored in RSULT is 1.5.

MIDRANGE
MIDRANGE

Description
The MIDRANGE (middle range) function returns a numeric value that is the arithmetic mean (average) of the values of the minimum argument and the maximum argument.

General Format
FUNCTION MIDRANGE ({num}...)
[num]
is a numeric argument.

Rules
1. The type of this function is numeric.
2. The returned value is the arithmetic mean of the greatest argument value and the least argument value. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions. (See Chapter 6, Procedure Division.)

3. The values of the arguments that are neither the greatest nor the least in value do not affect the value returned.

**Example**

COMPUTE RESULT = FUNCTION MIDRANGE (1, 2, 50, 4, 3).

The value returned and stored in RESULT (a numeric data item) is 25.5, which is the arithmetic mean of the greatest and least arguments; that is, the sum of 50 and 1 divided by 2.

---

**MIN**

**Description**

The MIN function returns the content of the argument that contains the minimum value.

**General Format**

FUNCTION MIN ({argument}...)

[argument]

is an alphabetic, alphanumeric, integer, or numeric argument.

**Rules**

1. The arguments must be all alphabetic, all alphanumeric, all integer, or all numeric, except that integer and numeric arguments can be mixed and alphabetic and alphanumeric arguments can be mixed.

2. The type of the function depends on the arguments, as follows:

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Function Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabetic and/or alphanumeric</td>
<td>Alphanumeric</td>
</tr>
<tr>
<td>Integer (all arguments)</td>
<td>Integer</td>
</tr>
<tr>
<td>Numeric (some arguments might be integer)</td>
<td>Numeric</td>
</tr>
</tbody>
</table>

3. The returned value consists of the contents of the argument having the least value, as determined by comparisons made according to the rules for simple conditions. (See Chapter 6, Procedure Division.)

4. If more than one argument has the same value, and that value is the minimum, the returned value consists of the contents of the leftmost of these arguments.

5. If there is only one argument, the returned value consists of the contents of that argument.

6. If the type of the function is alphanumeric, the size of the returned value is the same as the size of the argument selected as the minimum.

**Example**

COMPUTE ITEMC = (ITEMA + FUNCTION MIN (ITEMB, 10)).
Intrinsic Functions

If ITEMA and ITEMB both contain the value 1, this statement results in ITEMC having the value 2.

MOD

Description
The MOD function returns the value of argument-1 modulo argument-2.

General Format
FUNCTION MOD (argument-1 argument-2)

[argument-1]
is an integer argument.

[argument-2]
is an integer argument whose value cannot be 0.

Rules
1. The type of this function is integer.
2. The returned value is an integer value and is defined as the following:
   
   argument-1  –  (argument-2 * FUNCTION INTEGER (argument-1 / argument-2))
   
   (The INTEGER function returns the greatest integer value that is less than or equal to the argument. See
   INTEGER for more information.)

Example
COMPUTE RESULT = FUNCTION MOD (ARGUMENT-1, ARGUMENT-2).

ARGUMENT-1 and ARGUMENT-2 are numeric integer data items. Following are the expected results for some
values of ARGUMENT-1 and ARGUMENT-2:

<table>
<thead>
<tr>
<th>ARGUMENT-1</th>
<th>ARGUMENT-2</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>#11</td>
<td>#5</td>
<td>#1</td>
</tr>
<tr>
<td>-11</td>
<td>#5</td>
<td>#4</td>
</tr>
<tr>
<td>#11</td>
<td>-5</td>
<td>-4</td>
</tr>
<tr>
<td>-11</td>
<td>-5</td>
<td>-1</td>
</tr>
</tbody>
</table>

NUMVAL

Description
The NUMVAL function returns the numeric value represented by the character string specified by the argument.
Leading and trailing spaces are ignored.
General Format

FUNCTION NUMVAL (arg)

[arg]

is an alphanumeric argument whose content has one of the following two formats:

**Format 1**

[space] { + | - } [space] { digit [. digit]| . digit } [space]

**Format 2**

[space] { digit [. digit]| . digit } [space] { + | - | CR | DB } [space]

where space is a string of 0 or more spaces, and digit is a string of 1 to 18 digits.

**Rules**

1. The type of this function is numeric.
2. The total number of digits in the argument must not exceed 18.
3. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, a comma must be used in the argument rather than a decimal point.
4. The returned value is the numeric value represented by the argument.
5. The number of digits returned is 18.

**Examples**

1. COMPUTE RSULT = FUNCTION NUMVAL ("4540").
   The value returned and stored in RSULT (a numeric data item) is 4540.

2. MOVE "-123.49" TO OLD-ID.
   COMPUTE NEW-ID = 2 + FUNCTION NUMVAL (OLD-ID).

   OLD-ID is an alphanumeric data item, and NEW-ID is a numeric data item. The value returned by the function is the numeric value -123.49, which is added to 2, giving the sum -121.49, which is stored in NEW-ID.

**NUMVAL-C**

**Description**

The NUMVAL-C function returns the numeric value represented by the character string specified by the first argument. Any currency sign found in the character string and any commas preceding the decimal point are ignored in determining the result.

**General Format**

FUNCTION NUMVAL-C (arg-1 [arg-2])
Intrinsic Functions

[arg-1]

is an alphanumeric argument whose content has one of the following two formats:

**Format 1**

```
[space] + - [space] [cs] [space] digit [, [digit] ... [. [digit]] . digit [space]
```

**Format 2**

```
[space] [cs] [space] digit [, [digit] ... [. [digit]] . digit [space] + - CR DB [space]
```

where *space* is a string of 0 or more spaces, *cs* is a string of 1 or more characters specified by *arg-2*, and *digit* is a string of 1 or more digits.

[arg-2]

(if specified) is an alphanumeric argument.

### Rules

1. The type of this function is numeric.
2. The total number of digits in the first argument must not exceed 18.
3. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, the functions of the comma and decimal point in the first argument are reversed.
4. If the optional second argument is not specified, the character used for *cs* is the currency symbol specified for the program.
5. The returned value is the numeric value represented by the first argument.
6. The number of digits returned is 18.

### Example

```
COMPUTE RESULT = FUNCTION NUMVAL-C ("#1,000.00", ",").
```

The numeric value returned and stored in RESULT (a numeric data item) is 1000.

### ORD

**ORD**

**Description**

The ORD function returns an integer value that is the ordinal position of the argument in the collating sequence for the program. The lowest ordinal position is 1.

**General Format**

```
FUNCTION ORD (arg)
```

[ arg ]
is an alphabetic or alphanumeric argument one character in length.

**Rules**

1. The type of this function is integer.

2. The value returned is the ordinal position of the specified character in the program collating sequence. (See the information on the ALPHABET clause in Chapter 4, *Environment Division*.)

**Example**

```plaintext
COMPUTE POSITION = FUNCTION ORD (SINGLE-CHAR).
```

If SINGLE-CHAR (an alphabetic or alphanumeric data item) has the value "A", the integer representing the ordinal position of "A" in the program collating sequence (66 for native) is the value returned and stored in POSITION (a numeric integer data item). (The numeric representation of a character is not the same as its ordinal position. Numeric representation starts at 0, whereas ordinals start at 1. Thus, the ordinal value of a character is always 1 greater than its numeric value.)

**ORD-MAX**

**Description**

The ORD-MAX function returns a value that is the ordinal number of the argument that contains the maximum value.

**General Format**

```plaintext
FUNCTION ORD-MAX ([arg]...)
```

[arg]

is an alphabetic, alphanumeric, integer, or numeric argument.

**Rules**

1. The type of this function is integer.

2. The arguments must be all alphabetic, all alphanumeric, all integer, or all numeric, except that integer and numeric arguments can be mixed and alphabetic and alphanumeric arguments can be mixed.

3. The returned value is the ordinal number that corresponds to the position of the argument having the greatest value in the argument series.

4. The comparisons used to determine the greatest value are made according to the rules for simple conditions. (See Chapter 6, *Procedure Division*.)

5. If more than one argument has the same greatest value, the number returned corresponds to the position of the leftmost argument having that value.

6. If there is only one argument, the value returned is 1.

**Example**

```plaintext
COMPUTE RESULT = FUNCTION ORD-MAX (A, B, C).
```
A, B, and C are alphanumeric data items one character in length. If the value "A" is in A, "B" is in B, and "C" is in C, the value returned and stored in RSULT (a numeric data item) is 3, because the third argument, C, has the greatest value.

**ORD-MIN**

**Description**

The ORD-MIN function returns a value that is the ordinal number of the argument that contains the minimum value.

**General Format**

FUNCTION ORD-MIN ({arg}…)

[arg]

is an alphabetic, alphanumeric, integer, or numeric argument.

**Rules**

1. The type of this function is integer.
2. The arguments must be all alphabetic, all alphanumeric, all integer, or all numeric, except that integer and numeric arguments can be mixed and alphabetic and alphanumeric arguments can be mixed.
3. The returned value is the ordinal number that corresponds to the position of the argument having the least value in the argument series.
4. The comparisons used to determine the least value are made according to the rules for simple conditions. (See Chapter 6, Procedure Division.)
5. If more than one argument has the same least value, the number returned corresponds to the position of the leftmost argument having that value.
6. If there is only one argument, the value returned is 1.

**Example**

COMPUTE RSULT = FUNCTION ORD-MIN (A, B, C).

A, B, and C are alphanumeric data items one character in length. If the value "A" is in A, "B" is in B, and "C" is in C, the value returned and stored in RSULT (a numeric data item) is 1, because the first argument, A, has the least value.

**PRESENT-VALUE**

**Description**

The PRESENT-VALUE function returns a value that approximates the present value of a series of future period-end amounts at a discount rate. The discount rate is specified by the first argument, and the future period-end amount(s) by one or more subsequent arguments.
FUNCTION PRESENT-VALUE \{rate \{amt\}…\)

[rate]

is a numeric argument greater than -1 representing the discount rate.

[amt]

is a numeric argument representing a future period-end amount.

**Rules**

1. The type of this function is numeric.

2. The period-end amounts specified must be for periods of equal duration, and the discount rate must be the rate per period (for example, if each period is a year, then use the annual discount rate).

3. The returned value is an approximation of the summation of a series of calculations with each term in the following form:

\[ \frac{amt}{(1 + rate)^n} \]

There is one term for each occurrence of \(amt\). The exponent, \(n\), is incremented from 1 by 1 for each term in the series. If there are four arguments (rate, \(amt-1\), \(amt-2\), \(amt-3\)), the calculation is as follows:

\[
\text{present-value} = \frac{\text{amt-1}}{(1 + \text{rate})^1} + \frac{\text{amt-2}}{(1 + \text{rate})^2} + \frac{\text{amt-3}}{(1 + \text{rate})^3}
\]

**Example**

\[
\text{COMPUTE RESULT} = \text{FUNCTION PRESENT-VALUE (DISCOUNT-RATE, 2000)}.
\]

DISCOUNT-RATE and RESULT are numeric data items. If DISCOUNT-RATE has the value 0.08, the value returned and stored in RESULT is approximately 1851.85.

**RANDOM**

**Description**

The RANDOM function returns a numeric value that is a pseudo-random number from a rectangular distribution.

**General Format**

FUNCTION RANDOM [\{seed\}]

[seed]

is an optional integer argument with the value of 0 or a positive integer, used as the seed value to generate a sequence of pseudo-random numbers. The range of seed values that results in unique sequences of pseudo-random numbers is 0 through 2147483647.

**Rules**

1. The type of this function is numeric.
2. If the optional seed argument is not specified by the first reference to this function in the run unit, the seed value is 0.

3. If any subsequent reference to this function in the run unit does not specify the seed argument, the value returned is the next number in the current sequence of pseudo-random numbers.

4. If a subsequent reference to this function in the run unit does specify the seed argument, a new sequence of pseudo-random numbers is started.

5. The returned value is greater than or equal to 0 and less than 1.

6. For a given seed value, the sequence of pseudo-random numbers is always the same.

Example

```plaintext
COMPUTE RSULT-1 = FUNCTION RANDOM (12345).
.
.
.
COMPUTE RSULT-2 = FUNCTION RANDOM.
.
.
.
COMPUTE RSULT-3 = FUNCTION RANDOM (12345).
```

RSULT-1, RSULT-2, and RSULT-3 are numeric data items. Assuming the three sentences in the example are in the same run unit, the values returned and stored in RSULT-1 and RSULT-3 are the same. RSULT-2 has a different value consisting of the next number in the sequence that was started by the first reference to the function.

RANGE

Description

The RANGE function returns a value that is equal to the value of the maximum argument minus the value of the minimum argument.

General Format

```
FUNCTION RANGE({num}...)
```

[num]

is a numeric or integer argument.

Rules

1. The type of this function depends upon the argument types, as follows:

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Function Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer (all arguments)</td>
<td>Integer</td>
</tr>
<tr>
<td>Numeric (some arguments might be integer)</td>
<td>Numeric</td>
</tr>
</tbody>
</table>

2. The returned value is equal to the greatest value in the series of arguments minus the least value in the series.
3. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions. (See Chapter 6, Procedure Division.)

4. If only one argument is specified, the value returned is 0.

**Example**

```plaintext
COMPUTE RSULT = FUNCTION RANGE (4, 8, 10).
```

The value returned and stored in RSULT (a numeric integer data item) is 6.

**REM**

**Description**

The REM function returns a numeric value that is the remainder of the first argument divided by the second argument.

**General Format**

```plaintext
FUNCTION REM (arg-1 arg-2)
```

[arg-1]

is a numeric or integer argument.

[arg-2]

is a numeric or integer argument whose value cannot be 0.

**Rules**

1. The type of this function is numeric.

2. The returned value is the remainder of the first argument divided by the second argument, and is defined as the following expression:

   ```plaintext
   arg-1 - (arg-2 * FUNCTION INTEGER-PART (arg-1 / arg-2))
   ```

   (The INTEGER-PART function returns an integer that is the integer portion of its argument. See INTEGER-PART.)

**Examples**

1. ```plaintext
   COMPUTE RSULT = FUNCTION REM (3, 2).
   ```
   The value returned and stored in RSULT (a numeric data item) is 1.

2. ```plaintext
   COMPUTE RSULT = FUNCTION REM (4, 2).
   ```
   The value returned and stored in RSULT is 0.

**REVERSE**

REVERSE
Description

The REVERSE function returns a character string of exactly the same length as the argument and whose characters are exactly the same as those of the argument, except that they are in reverse order.

General Format

FUNCTION REVERSE (arg)

[arg]

is an alphabetic or alphanumeric argument at least one character in length.

Rules

1. The type of this function is alphanumeric.
2. If the argument is a character string of length n, the returned value is a character string of length n.
3. When 1 is less than or equal to j and j is less than or equal to n, the character in position j of the returned value is the character from position (n-j)+1 of the argument.

Example

MOVE FUNCTION REVERSE (STR) TO RESULT.

STR and RESULT are alphanumeric data items four characters in length. If STR contains the value "ABCD" then "DCBA" is the value returned and stored in RESULT.

If the value "AB" is moved to the four-character data item STR, then STR will actually contain "AB##" with two trailing spaces. Then the REVERSE function returns the value "##BA" with two leading spaces.

SIN

Description

The SIN function returns a numeric value that approximates the sine of an angle or arc, expressed in radians, that is specified by the argument.

General Format

FUNCTION SIN (angle)

[angle]

is a numeric argument having the value of the measurement in radians of an angle or arc.

Rules

1. The type of this function is numeric.
2. The returned value is the approximation of the sine of angle, and is greater than or equal to -1 and less than or equal to +1.
Example

COMPUTE SIN-RSLT = FUNCTION SIN (X).

If the value of X is 3, the approximate sine of an angle of 3 radians is moved to SIN-RSLT (a numeric data item).

SQRT

Description

The SQRT function returns a numeric value that approximates the square root of the argument.

General Format

FUNCTION SQRT (num)

[num]

is a numeric or integer argument whose value must be 0 or positive.

Rules

1. The type of this function is numeric.
2. The returned value is the absolute value of the approximation of the square root of the argument.

Example

COMPUTE RSULT = FUNCTION SQRT (NUM).

NUM and RSULT are numeric data items. If NUM has the value 4, the value returned and stored in RSULT is 2.

STANDARD-DEVIATION

Description

The STANDARD-DEVIATION function returns a numeric value that approximates the standard deviation of its arguments.

General Format

FUNCTION STANDARD-DEVIATION ({arg}...)

[arg]

is a numeric or integer argument.

Rules

1. The type of this function is numeric.
2. The returned value is the approximation of the standard deviation of the argument series.

3. The returned value is calculated as follows:
   a. The difference between each argument's value and the arithmetic mean (average) of the argument series is calculated and squared.
   b. The values obtained are then added together. This sum is divided by the number of values in the argument series.
   c. The square root of the quotient obtained is then calculated. The returned value is the absolute value of this square root.

4. If the argument series consists of only one value, the returned value is 0.

**Example**

```plaintext
COMPUTE RSULT = FUNCTION STANDARD-DEVIATION (A, B, C).
```

A, B, C, and RSULT are numeric data items. If A has the value 1, B has 2, and C has 12, the standard deviation of these values (approximately 4.96655) is returned and stored in RSULT.

**SUM**

**Description**

The SUM function returns a value that is the sum of the arguments.

**General Format**

```plaintext
FUNCTION SUM ({arg}…)
```

[arg]

is an integer or numeric argument.

**Rules**

1. The type of this function depends on the argument types, as follows:

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Function Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer (all arguments)</td>
<td>Integer</td>
</tr>
<tr>
<td>Numeric (some arguments might be integer)</td>
<td>Numeric</td>
</tr>
</tbody>
</table>

2. The returned value is the sum of the arguments.

**Examples**

1. ```plaintext
   COMPUTE RSULT = FUNCTION SUM (A, B, C).
   ```
   A, B, C, and RSULT are numeric or numeric integer data items. If A has the value +4, B -2, and C +1, the sum of +3 is the value returned and stored in RSULT.

2. ```plaintext
   COMPUTE TOTAL-OUT = 
   FUNCTION SUM(FUNCTION SQRT(X),
   ```
FUNCTION MOD(Y, Z),
A * B,
FUNCTION ACOS(1)).

This example shows functions used as arguments to another function. The data items are all numeric or numeric integer. The value returned and stored in TOTAL-OUT is the approximate value of the result of adding the values returned by the functions SQRT, MOD, and ACOS to another arithmetic expression, A * B.

3. The following example shows two arguments that are tables, with generic (ALL) subscripting, and a third argument that is a literal:

FUNCTION SUM(A(ALL), B(ALL, 2), 4)

The number of subscripts shows that A is a one-dimensional table and B is a two-dimensional table. If A has three occurrences, then A(ALL) is a set consisting of the elements A(1), A(2), and A(3). If B has two occurrences in its outer dimension, then B(ALL, 2) is a set consisting of the elements in B(1, 2) and B(2, 2).

If A has three elements altogether with the values 2 in A(1), 3 in A(2), and 3 in A(3), and if B has the values 9 in B(1, 2) and 3 in B(2, 2), then the value returned is 24—the sum of 2, 3, 3 (from table A), 9, 3 (from table B), and 4 (the third argument).

TAN

TAN

Description

The TAN function returns a numeric value that approximates the tangent of an angle or arc, expressed in radians, that is specified by the argument.

General Format

FUNCTION TAN (arg)

[arg]

is a numeric or integer argument.

Rules

1. The type of this function is numeric.

2. The returned value is the approximate tangent of the angle specified.

Example

COMPUTE TAN-RSLT = FUNCTION TAN (X).

X and TAN-RSLT are numeric data items. If the value of X is 3, the approximate tangent of an angle of 3 radians is moved to TAN-RSLT.

TEST-DATE-YYYYMMDD

TEST-DATE-YYYYMMDD

Description
The TEST-DATE-YYYYMMDD function tests whether a standard date in the form YYYYMMDD is a valid date in the Gregorian calendar.

**General Format**

FUNCTION TEST-DATE-YYYYMMDD ( arg )

[arg]

is an integer.

**Rules**

1. The type of this function is integer.
2. If the year is not within the range 1601 through 9999, the function returns a 1.
   
   Otherwise, if the month is not within the range 1 through 12, the function returns a 2.
   
   Otherwise, if the number of days is invalid for the given month, the function returns a 3.
   
   Otherwise, the function returns a 0 to indicate the date is a valid date in the form YYYYMMDD.

**Example**

IF FUNCTION TEST-DATE-YYYYMMDD (123456789) = 1
   DISPLAY "correct - invalid year (12345)".
IF FUNCTION TEST-DATE-YYYYMMDD (19952020) = 2
   DISPLAY "correct - invalid mm (20)".
IF FUNCTION TEST-DATE-YYYYMMDD (19950229) = 3
   DISPLAY "correct - invalid dd (29)".
IF FUNCTION TEST-DATE-YYYYMMDD (20040229) = 0
   DISPLAY "correct - valid YYYYMMDD".

**TEST-DAY-YYYYDDD**

**Description**

The TEST-DAY-YYYYDDD function tests whether a Julian date in the form YYYYDDD is a valid date in the Gregorian calendar.

**General Format**

FUNCTION TEST-DAY-YYYYDDD ( arg )

[arg]

is an integer.

**Rules**

1. The type of this function is integer.
2. If the year is not within the range 1601 through 9999, the function returns a 1.
Intrinsic Functions

Otherwise, if the number of days is invalid for the given year, the function returns a 2.

Otherwise, the function returns a 0 to indicate the date is a valid date in the form YYYYDDD.

Example

IF FUNCTION TEST-DAY-YYYYDDD (12345678) = 1
  DISPLAY "correct - invalid year (12345)".
IF FUNCTION TEST-DAY-YYYYDDD (1995366) = 2
  DISPLAY "correct - invalid ddd (366)".
IF FUNCTION TEST-DAY-YYYYDDD (2004366) = 0
  DISPLAY "correct - valid YYYYDDD".

UPPER-CASE

UPPER-CASE

Description

The UPPER-CASE function returns a character string that is the same length as the argument with each lowercase letter in the argument replaced by the corresponding uppercase letter.

General Format

FUNCTION UPPER-CASE (string)

[string]

is an alphabetic or alphanumeric argument at least one character in length.

Rules

1. The type of this function is alphanumeric.

2. The returned value is the same character string as the argument, except that each lowercase letter in the argument is replaced by the corresponding uppercase letter.

Examples

1. MOVE FUNCTION UPPER-CASE (STR) TO UC-STR.

   If STR (an alphanumeric data item six characters in length) contains the value "Autumn" the value returned and stored in UC-STR (also an alphanumeric data item six characters in length) is "AUTUMN"; if STR contains "FALL98" the value returned is unchanged ("FALL98").

2. ACCEPT NAME-FIELD.

   WRITE RECORD-OUT
     FROM FUNCTION UPPER-CASE(NAME-FIELD).

   The value in NAME-FIELD is made all-uppercase, unless it was already all-uppercase, in which case it is unchanged. Any nonalphabetic characters remain unchanged.

VARIANCE

VARIANCE
Description

The VARIANCE function returns a numeric value that approximates the variance of its arguments.

General Format

FUNCTION VARIANCE ((arg)...)  
[arg]

is an integer or numeric argument.

Rules

1. The type of this function is numeric.
2. The returned value is the approximation of the variance of the argument series, and is defined as the square of the standard deviation of the argument series. (For a definition of standard deviation, see the description of the STANDARD-DEVIATION function.)
3. If the argument series consists of only one value, the returned value is 0.

Examples

1. COMPUTE RESULT = FUNCTION VARIANCE (A).

   The value returned and stored in RESULT is 0, because there is only one argument.

2. COMPUTE RESULT = FUNCTION VARIANCE (A, B, C).

   If A has the value 1, B has 2, and C has 12, the value returned and stored in RESULT is approximately 24.667. This represents the variance, which is the square of the standard deviation of these arguments; the calculation is described in the description of the STANDARD-DEVIATION function. In the above examples, A, B, C, and RESULT are numeric data items.

WHEN-COMPILED

Description

The WHEN-COMPILED function returns the date and time the program was compiled.

General Format

FUNCTION WHEN-COMPILED

Rules

1. The type of this function is alphanumeric.
2. The returned value is the date and time of compilation of the source program that contains this function. If the program is a contained program, the returned value is the compilation date and time associated with the separately compiled program in which it is contained.
3. The returned value denotes the same time as the compilation date and time provided in the listing of the source program and in the generated object code for the source program. The representation differs, and the precision can differ, as shown in the second example.

4. The contents of the character positions returned, numbered from left to right, are as follows:

<table>
<thead>
<tr>
<th>Character Positions</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-4</td>
<td>Four numeric digits of the year in the Gregorian calendar.</td>
</tr>
<tr>
<td>5-6</td>
<td>Two numeric digits of the month of the year, in the range 01 through 12.</td>
</tr>
<tr>
<td>7-8</td>
<td>Two numeric digits of the day of the month, in the range 01 through 31.</td>
</tr>
<tr>
<td>9-10</td>
<td>Two numeric digits of the hours past midnight, in the range 00 through 23.</td>
</tr>
<tr>
<td>11-12</td>
<td>Two numeric digits of the minutes past the hour, in the range 00 through 59.</td>
</tr>
<tr>
<td>13-14</td>
<td>Two numeric digits of the seconds past the minute, in the range 00 through 59.</td>
</tr>
<tr>
<td>15-16</td>
<td>Two numeric digits of the hundredths of a second past the second, in the range 00 through 99.</td>
</tr>
<tr>
<td>17-21</td>
<td>The value 00000. (Reserved for future use.)</td>
</tr>
</tbody>
</table>

**Examples**

MOVE FUNCTION WHEN-COMPILED TO VERSION-STAMP.

The value returned and stored in VERSION-STAMP (an alphanumeric data item) is the date and time of the program's compilation.

199701101652313200000

This is a sample value returned by the WHEN-COMPILED function. Reading from left to right, it shows:

- The year, 1997
- The month, January
- The day of the month, the 10th
- The time of day, 16:52 (4:52 P.M.)
- The seconds, 31, and the hundredths of seconds, 32, after 16:52:31

This compilation date and time as shown on the compiler listing (which does not show hundredths of seconds) is as follows:

10-Jan-1997 16:52:31

**YEAR-TO-YYYY**

**YEAR-TO-YYYY**

**Description**

The YEAR-TO-YYYY function converts a two-digit year to a four-digit year. An optional second argument, when added to the current year (at the time the program executes), defines the ending year of a 100-year interval. This interval determines to what century the two-digit year belongs.
General Format

FUNCTION YEAR-TO-YYYY ( arg-1 [ arg-2 ] )

[arg-1]

is a nonnegative integer between 0 and 99.

[arg-2]

is an integer. Its value, when added to the current year, must be between 1700 and 9999. If it is omitted, the default value is 50.

Rules

1. The type of this function is integer.

2. The returned value is an integer representing a four-digit year calculated as follows:

\[
\text{max-year} = \text{current-yyyy} + \text{arg-2}
\]

\[
\text{if } \text{mod}(\text{max-year}, 100) \geq \text{arg-1}
\]

\[
\text{return } (\text{arg-1} + 100 \times \text{int}(\text{max-year} / 100))
\]

\[
\text{else}
\]

\[
\text{return } (\text{arg-1} + 100 \times \text{int}(\text{max-year} / 100) - 1)
\]

Example

IF FUNCTION YEAR-TO-YYYY (80, 50 ) = 1980
DISPLAY "correct".

IF FUNCTION YEAR-TO-YYYY (80, 100 ) = 2080
DISPLAY "correct".

IF FUNCTION YEAR-TO-YYYY (80, -100 ) = 1880
DISPLAY "correct".

YEAR-TO-YYYY implements a sliding window algorithm. To use it for a fixed window, you can specify arg-2 as follows:

(fixed-ending-year - function numval (function current-date (1:4)))

If fixed-ending-year is 2100, then for 1999 arg-2 has the value 101. If arg-1 is 50, the returned-value is 2050. If arg-1 is 99, the returned-value is 2099.
Chapter 8. Source Text Manipulation

Source programs can copy frequently used COBOL text from a Tru64 UNIX directory containing library files, an OpenVMS Librarian file, a COBOL library file, or (for OpenVMS) Oracle CDD/Repository. The COPY statement can include text without change, or it can change the text as it is copied into the source program.

The COPY statement REPLACING phrase changes text in the copying process. It matches arguments against the text to determine which text to replace. The matching procedure operates on text-words.

The REPLACE statement changes text in the source program. It matches source text to the pseudo-text specified in the REPLACE statement and changes the specified text when a match is detected.

8.1. Text-Word Definition Rules

A text-word is a character or sequence of characters in a COBOL library, source program, pseudo-text, or repository. It can be any of the following:

1. A literal, including the opening and closing quotation marks for nonnumeric literals
2. A hexadecimal literal, including the opening and closing delimiters
3. A separator other than:
   A space
   A pseudo-text delimiter
   The opening and closing quotation marks of a nonnumeric literal
4. Any other sequence of contiguous characters, bounded by separators, except:
   Comment lines
   Separators

Examples

These examples show how the compiler interprets COBOL text in terms of text-words. The rule letters refer to the text-word definition rules.

<table>
<thead>
<tr>
<th>Text</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE</td>
<td>One text-word (Rule 4).</td>
</tr>
<tr>
<td>MOVE ITEMA TO ITEMB</td>
<td>Four text-words.</td>
</tr>
<tr>
<td>MOVE ITEMA TO ITEMB.</td>
<td>Five text-words. The separator period is a text-word (Rule 3).</td>
</tr>
<tr>
<td>PIC S9(4)V9(6)</td>
<td>Nine text-words. Each parenthesis is a separator, and therefore a text-word. The nine text-words are PIC, S9, (, 4, ), V9, (, 6, and ).</td>
</tr>
<tr>
<td>“PIC S9(4)V9(6)”</td>
<td>One text-word (Rule 1).</td>
</tr>
<tr>
<td>X “4865784C6974”</td>
<td>One text-word (Rule 2).</td>
</tr>
<tr>
<td>ITEMA.</td>
<td>Two text-words. ITEMA and the separator period are text words.</td>
</tr>
<tr>
<td>==ITEMA. #==</td>
<td>Two text-words. The pseudo-text delimiters are not text-words (Rule 3). However, the separator period is a text-word.</td>
</tr>
<tr>
<td>==ITEMA.==</td>
<td>One text-word. The pseudo-text delimiters are not text-words. The punctuation character period is part</td>
</tr>
</tbody>
</table>
COPY

Function

The COPY statement includes text in a COBOL program.

General Formats

[text-name]

is the name of a COBOL library file available during compilation; or, if library-name is specified, is the name of a text record within the library file. (See Technical Notes.)

[library-name]

is the directory that contains library files on the Tru64 UNIX system; or, on OpenVMS, is the name of the OpenVMS Librarian library file that contains text-name. (See Technical Notes.)

[pseudo-text-1]

are text-matching arguments that the compiler compares against text-words in the library text.

[pseudo-text-2]

are replacement items that the compiler inserts into the source program.

[record-name (OpenVMS)]

is a partial or complete Oracle CDD/Repository pathname. It specifies the Oracle CDD/Repository record description to be copied into the source program. (See Technical Notes.)

Syntax Rules

1. A Format 1 COPY statement can be used anywhere that a character-string or separator (other than a closing quotation mark) can be used in a program.

2. On OpenVMS, a Format 2 COPY statement can appear only in the File, Working-Storage, or Linkage Sections.

3. A space must precede the word COPY.

4. The COPY statement must be terminated by the separator period.

5. pseudo-text-1 must contain at least one text-word.

6. pseudo-text-2 can contain zero, one, or more text-words.

7. word-1 or word-2 can be any COBOL word.

8. pseudo-text-1 must not consist entirely of a separator comma or a separator semicolon.
General Rules

Format 1

1. On Tru64 UNIX, when both text-name and library-name are specified, library-name refers to the directory containing library files; text-name identifies a specific file within the directory.

2. On OpenVMS, when both text-name and library-name are specified, library-name refers to an OpenVMS Librarian library file; text-name identifies a text record within the library file.

3. When only text-name is used, it identifies a file that contains library text.

4. Library text must follow the source reference format rules. Library text and source program text formats must be the same; that is, both must be ANSI format, or both must be terminal format.

5. On Tru64 UNIX, the COPY statement references source text from a directory containing library files or from a COBOL library file.

Format 2 (OpenVMS)

6. record-name refers to a record description stored in Oracle CDD/Repository.

7. The compiler translates the record description associated with record-name to COBOL source text. If the source program containing the COPY statement is in terminal format, the translated record description is in terminal format; otherwise, the record description is translated to ANSI format.

Both Formats

8. On OpenVMS, the COPY statement references source text from an OpenVMS Librarian file, a COBOL library file, or the Oracle CDD/Repository.

9. The compiler evaluates the COBOL source program after processing all COPY statements.

10. The COPY statement does not change the original source program text file.

11. The COPY statement causes the compiler to copy the source text associated with text-name into the program. The source text logically replaces the COPY statement, beginning with the word COPY and ending with the punctuation character period (inclusive).

12. If there is no REPLACING phrase, the compiler copies the source text without modification.

13. If there is a REPLACING phrase, the compiler changes the source text as it copies it. The compiler replaces each successfully matched occurrence of a text-matching argument in the source text by the corresponding replacement item.

14. For the purposes of matching, the compiler treats each text-matching argument as pseudo-text that contains identifier-1, word-1, or literal-1.

15. The comparison operation starts with the leftmost source text text-word and the first text-matching argument. The compiler compares the entire text-matching argument to an equivalent number of consecutive source text text-words.

16. A text-matching argument matches the source text only if the ordered sequence of text-words that forms the text-matching argument is equal, character for character, to the ordered sequence of source text text-words.

In the matching operation, the compiler treats each occurrence or combination of the following items in source text as a single space:

- Separator comma
- Separator semicolon
Source Text Manipulation

- A sequence of one or more separator spaces
- A blank line
- A comment line

17. If no match occurs, the compiler repeats the comparison with each successive text-matching argument in the REPLACING phrase until either:
   - A match occurs.
   - There are no more text-matching arguments.

18. If no match occurs after the compiler compares all text-matching arguments, the compiler copies the leftmost source text text-word into the source program. The next source text text-word then becomes the leftmost text-word for the next cycle. The comparison cycle resumes with the first text-matching argument in the REPLACING phrase.

19. If a match occurs between a text-matching argument and the source text, the compiler inserts the replacement item into the source program. The source text-word immediately after the rightmost replaced text-word then becomes the leftmost text-word for the next cycle. The comparison cycle resumes with the first text-matching argument in the REPLACING phrase.

20. The comparison cycle continues until the rightmost text-word in the source text has been either:
   - Matched and replaced
   - Used as the leftmost library text-word in a comparison cycle

21. The rules for Reference Format determine the sequence of text-words in the source text and the text-matching arguments.

22. When the compiler inserts *pseudo-text-2* into the source program, any comment lines or blank lines within *pseudo-text-2* are inserted without modification. (See Example 5.)

23. The compiler copies any comment lines and blank lines in the source text into the source program unchanged (see Example 1). However, the compiler does not copy a comment line or blank line from the source text if it is in the sequence of text-words that matches the text-matching argument.

24. The resultant source program cannot contain a COPY statement after the compiler processes a COPY statement.
   - Text copied from a source text cannot contain a COPY statement unless the replacement operation changes the word COPY in the resultant source text.
   - The replacement item in the REPLACING phrase must not insert a COPY statement into the source text.

25. The compiler cannot determine the syntactic correctness of source text, or the source program, until all COPY statements are processed.

26. When the compiler copies a text-word from the source text, it places it in the source program beginning in the same area as in the source text. That is, a text-word that begins in Area A in the source text begins in Area A of the source program after the copy operation. Similarly, a text-word that begins in Area B in the source text begins somewhere in Area B of the source program.

27. When the compiler inserts a text-word from *pseudo-text-2*, it places it in the source program beginning in the same area as in *pseudo-text-2*.

28. When the compiler inserts text from *identifier-2, literal-2, or word-2*, it places the first text-word in the source program beginning in the same area as the leftmost library text-word that matches the argument. It places all other replacement text-words in the source program beginning in the same area as they appear in the COPY statement.
29. Pseudo-text insertion can change parts of a single character-string. An unmatched text-word and a replaced text-word can combine to form a character-string. For example, the COPY statement can replace part of a PICTURE character-string. (See Example 3.)

30. Conditional compilation lines are permitted within the library text and pseudo-text. Text-words within a conditional compilation line participate in the matching process as if the indicator area character of the line on which they began was not present. A conditional compilation line is specified within pseudo-text if it begins in the source program after the opening pseudo-text delimiter, but before the matching closing pseudo-text delimiter.

31. The resultant text can occur on conditional compilation lines according to the following precedence rules:
   • If a COPY statement begins on a conditional compilation line, each line of the resulting text appears on the same kind of line.
   • If a library text-word that is not involved in a match begins on a conditional compilation line, it appears in resulting text on the same kind of line.
   • If the first library text-word that is involved in a match begins on a conditional compilation line, the identifier-2, literal-2, word-2, or pseudo-text-2 that replaces the first library text-word appears on the same kind of line.
   • If text-words within pseudo-text-2 begin on a conditional compilation line, resulting text appears on the same kind of line.

Technical Notes

Format 1

1. If there is a library-name phrase, text-name is tr-name.
   
   On OpenVMS systems, tr-name is defined as a user-defined name or nonnumeric literal that matches the name of a text record in library-name.

2. 

3. On Alpha and I64 systems, the COBOL command-line qualifier /INCLUDE (or -include) can be used at compile time to set up a search list for the COPY command. Assume that the following conditions are all true:
   • library-name is not used.
   • text-name does not include a directory specification.
   • /INCLUDE is specified.
   
   Then the compiler searches for the text-name file in the following order:
   a. The current working directory when the compiler is invoked
   b. The directory specified after /INCLUDE (or -include)
      
      If more than one directory is specified, they are searched in left to right order.
      
      On Alpha and I64 systems, if library-name is used, and library-name does not include a directory specification, the /INCLUDE qualifier causes a search for a .TLB file, in the same search order.
      
      The first file encountered that matches the text-name terminates the search.
      
      On Tru64 UNIX, /INCLUDE (or -include) can be used to set up a search list for a text-name without a directory specification only if library-name is not specified.
      
      If the default, /NOINCLUDE, is in effect, a file without a directory specification is searched for only in the current default directory at compile time.
The pathname(s) specified with /INCLUDE can be relative or absolute directory specifications, logical names on OpenVMS Alpha or I64, or environment variables on Tru64 UNIX.

4. If f-name or tr-name is not a literal, the compiler translates hyphens in the word to underscore characters and treats it as if it were enclosed in quotation marks.

5. When the COPY statement executes, the I/O system:
   - Removes leading and trailing spaces and tab characters from f-name and from tr-name
   - Translates lowercase letters to uppercase in f-name and in tr-name

6. The default file type for text-name, when text-name is f-name, is LIB. For example, COPY CUSTFILE becomes COPY CUSTFILE.LIB.

7. On OpenVMS, the default file type for library-name is TLB. For example, COPY "ACCOUNTS" OF CUSTLIB becomes COPY "ACCOUNTS" OF CUSTLIB.TLB.

8. On all platforms, file names must conform to the rules of the operating system where compilation occurs. For example:
   - On Tru64 UNIX systems:
     COPY "/usr/proj/empl/empl_file".
     COPY "empl_file" IN "/usr/proj/empl".
   - On OpenVMS systems:
     COPY "COPYDIR:EMPL_FILE.LIB".
     COPY "EMPL_FILE" IN "EMPLLIB.TLB".

Format 2 (OpenVMS)

11. record-name can be a nonnumeric literal or COBOL word formed according to the rules for user-defined names. It represents a complete or partial Oracle CDD/Repository pathname specifying the Oracle CDD/Repository record description to be copied into the source program. If record-name is not a literal, the compiler translates hyphens in the COBOL word to underline characters.

The resultant pathname must conform to all rules for forming Oracle CDD/Repository pathnames.

12. Table 8.1, "Oracle CDD/Repository Data Types and VSI COBOL Equivalents (OpenVMS)" shows the representation of Oracle CDD/Repository data types in the VSI COBOL compiler. It lists the data types that can be specified using CDO with the corresponding COBOL data item picture. Note that COBOL does not have an equivalent specification for some data types.

<table>
<thead>
<tr>
<th>Oracle CDD/Repository Data Type</th>
<th>VSI COBOL Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT l</td>
<td>No equivalent</td>
</tr>
<tr>
<td>SIGNED BYTE l s</td>
<td>No equivalent</td>
</tr>
<tr>
<td>UNSIGNED BYTE l s</td>
<td>No equivalent</td>
</tr>
<tr>
<td>D_FLOATING s s</td>
<td>COMP-2 (with /FLOAT=D_FLOAT)</td>
</tr>
<tr>
<td>D_FLOATING COMPLEX s s</td>
<td>No equivalent</td>
</tr>
</tbody>
</table>

1 The total number of digits in the item.
2 The decimal offset to l.
<table>
<thead>
<tr>
<th>Oracle CDD/Repository Data Type</th>
<th>VSI COBOL Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>No exact equivalent</td>
</tr>
<tr>
<td>F_FLOATING s</td>
<td>COMP-1 (with /FLOAT=D_FLOAT or /FLOAT=G_FLOAT)</td>
</tr>
<tr>
<td>F_FLOATING COMPLEX s</td>
<td>No equivalent</td>
</tr>
<tr>
<td>G_FLOATING s</td>
<td>COMP-2 (with /FLOAT=G_FLOAT) on Alpha, I64; no equivalent on VAX</td>
</tr>
<tr>
<td>G_FLOATING COMPLEX s</td>
<td>No equivalent</td>
</tr>
<tr>
<td>H_FLOATING s</td>
<td>No equivalent</td>
</tr>
<tr>
<td>H_FLOATING COMPLEX s</td>
<td>No equivalent</td>
</tr>
<tr>
<td>IEEE S_FLOATING</td>
<td>COMP-1 (with /FLOAT=IEEE_FLOAT) on Alpha, I64; no equivalent on VAX</td>
</tr>
<tr>
<td>IEEE T_FLOATING</td>
<td>COMP-2 (with /FLOAT=IEEE_FLOAT) on Alpha, I64; no equivalent on VAX</td>
</tr>
<tr>
<td>SIGNED LONGWORD l s</td>
<td>S9 (9) COMP</td>
</tr>
<tr>
<td>UNSIGNED LONGWORD l s</td>
<td>No exact equivalent</td>
</tr>
<tr>
<td>UNSIGNED NUMERIC l s</td>
<td>9 (m)V9 (n)</td>
</tr>
<tr>
<td>SIGNED NUMERIC LEFT SEPARATE l s</td>
<td>S9 (m)V9 (n) LEADING SEPARATE</td>
</tr>
<tr>
<td>SIGNED NUMERIC LEFT OVERPUNCHED l s</td>
<td>S9 (m)V9 (n) LEADING</td>
</tr>
<tr>
<td>SIGNED NUMERIC RIGHT SEPARATE l s</td>
<td>S9 (m)V9 (n) TRAILING SEPARATE</td>
</tr>
<tr>
<td>SIGNED NUMERIC RIGHT OVERPUNCHED l s</td>
<td>S9 (m)V9 (n) TRAILING</td>
</tr>
<tr>
<td>SIGNED OCTAWORD l s</td>
<td>S9 (31) COMP on Alpha, I64; no equivalent on VAX</td>
</tr>
<tr>
<td>UNSIGNED OCTAWORD l s</td>
<td>No exact equivalent</td>
</tr>
<tr>
<td>PACKED NUMERIC l s</td>
<td>S9 (m)V9 (n) COMP-3</td>
</tr>
<tr>
<td>SIGNED QUADWORD l s</td>
<td>S9(18) COMP</td>
</tr>
<tr>
<td>UNSIGNED QUADWORD l s</td>
<td>No exact equivalent</td>
</tr>
<tr>
<td>TEXT m CHARACTERS</td>
<td>X (m)</td>
</tr>
<tr>
<td>UNSPECIFIED m BYTES</td>
<td>X (m)</td>
</tr>
<tr>
<td>VARYING STRING m CHARACTERS</td>
<td>No equivalent</td>
</tr>
<tr>
<td>VIRTUAL FIELD</td>
<td>Ignored</td>
</tr>
<tr>
<td>SIGNED WORD l s</td>
<td>S9 (4) COMP</td>
</tr>
<tr>
<td>UNSIGNED WORD l s</td>
<td>No exact equivalent</td>
</tr>
<tr>
<td>POINTER</td>
<td>POINTER</td>
</tr>
<tr>
<td>SEGMENTED STRING</td>
<td>No equivalent</td>
</tr>
<tr>
<td>ZONED</td>
<td>No equivalent</td>
</tr>
</tbody>
</table>

1 The total number of digits in the item.
2 The decimal offset to l.

1 COBOL has no equivalent for this data type. A warning diagnostic will be issued for such an item that is part of a record description entry. The compiler will treat that item as if it had been specified as an alphanumeric data item that occupies that same number of bytes.
2 COBOL has no exact equivalent for this data type. A warning diagnostic will be issued for such an item that is part of a record description entry. The compiler will treat that item as if it had been specified as PIC S9(11)V9(7) COMP. (This gives the item units of seconds.)
3 COBOL has no exact equivalent for this data type. A warning diagnostic will be issued for such an item that is part of a record description entry. The compiler will treat that item as if it had been specified as the corresponding unsigned COMP data type.
4 The VSI COBOL compiler ignores this data item and all its phrases.
The method for describing the assumed decimal point is different in the two products. In a COBOL picture, the
decimal position is directly indicated by the symbol V or implied by the symbol P. In CDO, scaled numbers
are specified by two integers: (1) the first integer represents the total number of decimal digits that the item
represents, and (2) the second integer represents the decimal offset to the first integer. These are indicated
in Table 8.1, “Oracle CDD/Repository Data Types and VSI COBOL Equivalents (OpenVMS)” by l and s,
respectively.

For example, the COBOL data item described by PIC 9(4)V99 is equivalent to the CDO entry UNSIGNED
NUMERIC 6 DIGITS SCALE -2. Similarly, the CDO entry SIGNED NUMERIC LEFT SEPARATE
NUMERIC 6 DIGITS SCALE 2 is equivalent to the COBOL description PIC S9(6)PP SIGN IS LEADING
SEPARATE. You can also represent digits to the right of the decimal point in CDO with the FRACTIONS
phrase. For example, instead of UNSIGNED NUMERIC 6 DIGITS SCALE -2, you can also use UNSIGNED
NUMERIC 6 DIGITS 2 FRACTIONS.

One of the primary goals of Oracle CDD/Repository is to describe data in such a way that data definitions
can be shared among many different processors. Many languages have different semantic interpretations for
the same physical data. Record descriptions in Oracle CDD/Repository must be able to describe the physical
characteristics of data unambiguously. In other words, the logical view of the data must be separated from the
physical description if different processors are to access the same record description.

VSI COBOL expects numeric literals and PICTURE character-strings to be obtained from Oracle CDD/
Repository in standard representation. Whether or not a particular COBOL source program uses the DECIMAL-
POINT IS COMMA clause or the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph, the record
description that was stored in Oracle CDD/Repository must have used the period (.) to represent the decimal
point in numeric literals and PICTURE character-strings, the comma (,) to represent the comma in PICTURE
class-character-strings, and the currency symbol ($) to represent the currency symbol in PICTURE character-strings.

When the COBOL source program contains the DECIMAL-POINT IS COMMA clause, the VSI COBOL
compiler substitutes commas for decimal points in numeric literals and PICTURE character-strings obtained
from Oracle CDD/Repository. It substitutes decimal points for commas in PICTURE character-strings obtained
from Oracle CDD/Repository.

When the COBOL source program contains the CURRENCY SIGN clause, the VSI COBOL
compiler substitutes the currency symbol for the currency sign in PICTURE character-strings obtained from Oracle CDD/
Repository.

Additional References

• Section 1.3: Source Reference Format
• Oracle CDD/Repository documentation set

Examples Using Format 1

The examples that follow copy library text from two library files:

• Contents of CUSTFILE.LIB:

```cobol
01 [Tab]CUSTOMER-REC.
   [Tab]  03   CUST-REC-KEY[Tab]PIC X(03) VALUE "KEY".
   [Tab]  03   CUST-ADDRESS.
      [Tab]   05   CUST-CITY[Tab]PIC X(20).
      [Tab]   05   CUST-STATE[Tab]PIC XX.
      [Tab]   05   CUST-ZIP[Tab]PIC 9(5).
* THE COMPILER IGNORES COMMENT LINES AND BLANK LINES
```

* FOR MATCHING PURPOSES
• Contents of CPROC01.LIB:

[Tab]ADD CUST-ORDER-AMT (X) TO TOTAL-ORDERS.
[Tab]COMPUTE AVERAGE-ORDER = (TOTAL-ORDERS - CANCELED-ORDERS)
[Tab]  / NUMBER-ORDERS.
[Tab]MOVE CUST-REC-KEY
[Tab]  OF CUSTOMER-REC TO CUST-ID (X).
[Tab]MOVE CUST-REC-KEY
[Tab]  OF KEY-HOLD TO NEW-KEY.

In the following examples, the original source program text is shown in lowercase text. The text that is copied is shown in uppercase. (The messages in these examples are in OpenVMS Alpha and I64 format.)

Example 8.1. “COPY with No REPLACING Phrase” shows the results of a COPY statement with no REPLACING phrase. The compiler copies the library text without change. In this example, syntax errors result from invalid library text.

Example 8.1. COPY with No REPLACING Phrase

Example 8.2. Replacing a Word with a Literal shows the results of replacing a word (“xyz”) by a literal (6).
Example 8.3, “Replacing a Word by a Literal and Pseudo-Text by Pseudo-Text” shows the results of replacing a word (“xyz”) by a literal (6), and pseudo-text by pseudo-text. The compiler recognizes R as a text-word because parentheses enclose it. The other R characters are not text-words; they are part of other text-words.

Example 8.3. Replacing a Word by a Literal and Pseudo-Text by Pseudo-Text

```cobol
39 copy custfile replacing xyz by 6, ==r== by ==4==.
```

Example 8.4, “Matching a Nonnumeric Literal” shows the results of matching a nonnumeric literal. The opening and closing quotation marks are part of the text-word.

Example 8.4. Matching a Nonnumeric Literal

```cobol
129 copy custfile replacing xyz by 6, "==r==" by "==4==".
```

%COBOL-F-PICREPEAT 178, (1) Invalid repetition factor
Example 8.5, “Multiple-Line Pseudo-Text Replacement Item” shows the results of a multiple-line pseudo-text replacement item. The replacement item starts after the pseudo-text delimiter on line 167 and ends before the delimiter on line 169. The continuation area on the new line (172) contains the same characters as line 168 in the pseudo-text replacement item. This example is not a recommended use of the COPY statement. It only shows the mechanics of the statement.

Example 8.5. Multiple-Line Pseudo-Text Replacement Item

```
166 copy custfile replacing xyz by 6, ==r== by ==4==
167   "KEY" by =="abc".
168  * cust-number is a new field
169    03 cust-number pic 9(8)==.
L 170  01 CUSTOMER-REC.
LR 171    03 CUST-REC-KEY   PIC X(03) VALUE "abc".
LR 172  * cust-number is a new field
LR 173    03 cust-number pic 9(8).
L 174    03 CUST-NAME   PIC X(25).
L 175    03 CUST-ADDRESS.
L 176     05 CUST-CUST-STREET PIC X(20).
L 177     05 CUST-CITY PIC X(20).
L 178     05 CUST-STATE PIC XX.
L 179     05 CUST-ZIP PIC 9(5).
L 180  * THE COMPILER IGNORES COMMENT LINES AND BLANK LINES
L 181
L 182  * FOR MATCHING PURPOSES
LR 183    03 CUST-ORDERS OCCURS 6 TIMES.
L 184     05 CUST-ORDER   PIC 9(6).
L 185     05 CUST-ORDER-DATE PIC 9(6).
LR 186     05 CUST-ORDER-AMT PIC 9(4)V99.
```

Example 8.6, “Matching Pseudo-Text That Includes Separators” shows the results of matching pseudo-text that includes separators.

The replacement phrase in line 210 fails to match the library text in line 212. The text-matching argument contains one text-word: the 13 characters beginning with c and ending with a period (.). The period is not a separator period, because it is not followed by a space. This argument fails to match the two text-words on line 212. The two text-words are: (1) CUSTOMER-REC and (2) the separator period.

The replacement phrase in line 211 replaces library text on line 215. The text-matching argument contains the same two text-words that are in the library text: (1) CUST-ADDRESS and (2) the separator period.

Example 8.6. Matching Pseudo-Text That Includes Separators

```
209 copy custfile replacing xyz by 6, ==r== by ==4==
210  ==customer-rec== by ==record-a==
211  ==cust-address. == by ==customer-address.==.
L 212  01 CUSTOMER-REC.
L 213    03 CUST-REC-KEY   PIC X(03) VALUE "KEY".
L 214    03 CUST-NAME   PIC X(25).
LR 215    03 customer-address.
L 216     05 CUST-CUST-STREET PIC X(20).
L 217     05 CUST-CITY PIC X(20).
L 218     05 CUST-STATE PIC XX.
L 219     05 CUST-ZIP PIC 9(5).
L 220  * THE COMPILER IGNORES COMMENT LINES AND BLANK LINES
L 221
L 222  * FOR MATCHING PURPOSES
LR 223    03 CUST-ORDERS OCCURS 6 TIMES.
L 224     05 CUST-ORDER   PIC 9(6).
```
Examples Using Format 2 (OpenVMS)

Figure 8.1, “Hierarchical Repository Structure (OpenVMS)” represents a hierarchical repository structure for Examples 8.7, 8.8, and 8.9. It contains one repository directory and two repository objects.

Figure 8.1. Hierarchical Repository Structure (OpenVMS)

In Figure 8.1, “Hierarchical Repository Structure (OpenVMS)”, the repository is named SALES (USA and GERMANY are not used). ANCHOR is the starting directory for the full repository pathname. Repository directories are analogous to OpenVMS Alpha and I64 subdirectories. They catalog other repository directories or repository objects, and they are labeled by the paths through the hierarchy that lead to them.

The repository objects are named PAYROLL and INVENTORY. These objects are the named record descriptions stored in Oracle CDD/Repository, and they form the end-points of the repository hierarchy branches. The examples that follow copy these record descriptions.

The full repository pathname provides a unique designation for every directory and object in Oracle CDD/Repository hierarchy. It traces the paths from ANCHOR to the directory or object.

For information on how to create and maintain a hierarchical structure in Oracle CDD/Repository, refer to the Oracle CDD/Repository documentation set.

Note
Not all Oracle CDD/Repository data types are valid VSI COBOL data types. See the Technical Notes.

Example 8.7, “Command File That Creates Oracle CDD/Repository Directories and Objects in Figure 8-1 (OpenVMS)” shows how to use a command file to create the repository directories and objects shown in Figure 8.1, “Hierarchical Repository Structure (OpenVMS)” using CDO.

Example 8.7. Command File That Creates Oracle CDD/Repository Directories and Objects in Figure 8-1 (OpenVMS)

```csh
define field name
datatype is text
size 30.
define field address
datatype is text
size is 40.
define field salesman_id
datatypes is text
size is 5.
define record salesman.
  name.
  address.
  salesman_id.
end record.
define field ytd_sales
datatype is right overpunched numeric
size is 11 digits
scale -2.
define field ytd_commission
datatype is right overpunched numeric
```
define field curr_month_sales
datatype is right overpunched numeric
size is 11 digits
scale -2.
define field curr_month_commission
datatype is right overpunched numeric
size is 11 digits
scale -2.
define field curr_week_sales
datatype is right overpunched numeric
size is 11 digits
scale -2.
define field curr_week_commission
datatype is right overpunched numeric
size is 11 digits
scale -2.
define record payroll_record.
salesman.
ytd_sales.
ytd_commission.
curr_month_sales.
curr_month_commission.
curr_week_sales.
curr_week_commission.
end record.
define field part_number
datatype is right overpunched numeric
size is 6 digits.
define field quantity_on_hand
datatype is right overpunched numeric
size is 9 digits.
define field quantity_on_order
datatype is right overpunched numeric
size is 9 digits.
define field retail_price
datatype is right overpunched numeric
size is 8 digits
scale -2.
define field wholesale_price
datatype is right overpunched numeric
size is 8 digits
scale -2.
define field supplier
datatype is text
size is 5 characters.
define record inventory_record.
part_number.
quantity_on_hand.
quantity_on_order.
retail_price.
wholesale_price.
supplier.
end record.

Example 8.8, “Using a Logical Name in a COPY Statement (OpenVMS)” shows the results of copying
the repository object PAYROLL in Figure 8.1, “Hierarchical Repository Structure (OpenVMS)”. The program
defines the logical name payroll to be equivalent to the full Oracle CDD/Repository pathname DEVICE:[DIRECTORY.ANCHOR]. Line 27 of the program shows the DCL command used to define the logical name and line 30 contains the COPY FROM DICTIONARY statement.

On OpenVMS Alpha and I64 systems, the COPY statement produces lines 31 to 44 in your program listing if you include the /COPY_LIST compiler option. Line 32 is the resulting full Oracle CDD/Repository pathname used by the compiler. Lines 31 and 33 are separator comment lines. Lines 34 to 44 are the COBOL compiler-translated record description entries taken from the PAYROLL repository object in Oracle CDD/Repository.

Example 8.8. Using a Logical Name in a COPY Statement (OpenVMS)

```
1 IDENTIFICATION DIVISION.
2 PROGRAM-ID. TEST-CDD.
3 *
4 Copy from CDD/Repository
5 * FILE SECTION
6 Records: PERSONNEL
7 INVENTORY
8 PAYROLL
9 *
10 WORKING-STORAGE SECTION
11 Records: SYDNEY
12 MAPLE
13 FRENCH
14 *
15 ENVIRONMENT DIVISION.
16 INPUT-OUTPUT SECTION.
17 FILE-CONTROL.
18 SELECT SALES-CDD-FILE
19 ASSIGN TO "CDD.TMP".
20 DATA DIVISION.
21 FILE SECTION.
22 FD SALES-CDD-FILE.
23 *
24 To create a logical name entry for the repository
25 object
26 *
27 $ DEFINE PAYROLL_RECORD "DEVICE:
[DIRECTORY.ANCHOR]SALES.PAYROLL"
28 *
29 *
30 COPY PAYROLL FROM DICTIONARY.
31 *
32 _DEVICE:[DIRECTORY.ANCHOR]PAYROLL_RECORD
33 *
34 01 PAYROLL_RECORD.
35 02 SALESMAN.
36 03 NAME PIC X(30).
37 03 ADDRESS PIC X(40).
38 03 SALESMAN_ID PIC X(5).
39 02 YTD_SALES PIC S9(9)V9(2) SIGN TRAILING.
40 02 YTD_COMMISSION PIC S9(9)V9(2) SIGN TRAILING.
41 02 CURR_MONTH_SALES PIC S9(9)V9(2) SIGN TRAILING.
42 02 CURR_MONTH_COMMISSION PIC S9(9)V9(2) SIGN TRAILING.
43 02 CURR_WEEK_SALES PIC S9(9)V9(2) SIGN TRAILING.
44 02 CURR_WEEK_COMMISSION PIC S9(9)V9(2) SIGN TRAILING.
```
Example 8.9, “Using a Full Pathname in a COPY Statement (OpenVMS)” shows the results of copying a repository object INVENTORY by specifying its full Oracle CDD/Repository pathname.

In Example 8.9, “Using a Full Pathname in a COPY Statement (OpenVMS)”, line 44 contains the COPY FROM DICTIONARY statement. On OpenVMS Alpha and I64 systems, this COPY statement produces lines 45 to 54 in your program listing if you include the /COPY_LIST compiler option. Line 46 is the resulting full Oracle CDD/Repository pathname used by the compiler. Lines 45 and 47 are separator comment lines. Lines 48 to 54 are the compiler-translated record description entries taken from the inventory repository object in Oracle CDD/Repository.

Example 8.9. Using a Full Pathname in a COPY Statement (OpenVMS)

```
COPY "DEVICE:[DIRECTORY.ANCHOR]INVENTORY_RECORD" FROM DICTIONARY.
```

Figure 8.2, “Nonhierarchical Repository Structure (OpenVMS)” shows a nonhierarchical repository structure. In this example, fields NAME and ADDRESS are used by both the EMPLOYEE-RECORD and the CUSTOMER-RECORD. As such, they are defined in a separate directory (COMMON_FIELD_DEFINITIONS). The fields PART and PART_NUMBER are used exclusively by the INVENTORY_RECORD. As such, they are defined in the INVENTORY directory. This functionality is only available in CDO formatted repositories.

Figure 8.2. Nonhierarchical Repository Structure (OpenVMS)

Example 8.10, “Command File That Creates Oracle CDD/Repository Directories and Objects in Figure 8-2 (OpenVMS)” shows how to use a CDO command file to create the directories and objects shown in Figure 8.2, “Nonhierarchical Repository Structure (OpenVMS)” using CDO. The CDO file is executed from within CDO using the following command:

```
$ REPOSITORY
CDO>@FILENAME.CDO
```
Example 8.10. Command File That Creates Oracle CDD/Repository Directories and Objects in Figure 8-2 (OpenVMS)

DEFINE DICTIONARY DEVICE:[DIRECTORY.ANCHOR].
SET DEFAULT DEVICE:[DIRECTORY.ANCHOR]
DEFINE DIRECTORY EMPLOYEE.
DEFINE DIRECTORY CUSTOMER.
DEFINE DIRECTORY INVENTORY.
DEFINE DIRECTORY COMMON_FIELD_DEFINITIONS.
SET DEFAULT DEVICE:[DIRECTORY.ANCHOR]COMMON_FIELD_DEFINITIONS
DEFINE FIELD NAME DATATYPE IS TEXT SIZE IS 25 CHARACTERS.
DEFINE FIELD ADDRESS DATATYPE IS TEXT SIZE IS 47 CHARACTERS.
SET DEFAULT DEVICE:[DIRECTORY.ANCHOR]EMPLOYEE
DEFINE FIELD DATE_OF_HIRE DATATYPE IS UNSIGNED NUMERIC SIZE IS 8 DIGITS.
DEFINE FIELD SEX DATATYPE IS TEXT SIZE IS 1 CHARACTER.
DEFINE FIELD DEPENDENTS DATATYPE IS UNSIGNED NUMERIC SIZE IS 2 DIGITS.
DEFINE RECORD EMPLOYEE_RECORD.
[DIRECTORY.ANCHOR]COMMON_FIELD_DEFINITIONS.NAME.
[DIRECTORY.ANCHOR]COMMON_FIELD_DEFINITIONS.ADDRESS.
DATE_OF_HIRE.
SEX.
DEPENDENTS.
END RECORD.
SET DEFAULT DEVICE:[DIRECTORY.ANCHOR]CUSTOMER
DEFINE FIELD BUSINESS_TYPE DATATYPE IS TEXT SIZE IS 25 CHARACTERS.
DEFINE FIELD CONTACT_PERSON DATATYPE IS TEXT SIZE IS 25 CHARACTERS.
DEFINE RECORD CUSTOMER_RECORD.
[DIRECTORY.ANCHOR]COMMON_FIELD_DEFINITIONS.NAME.
[DIRECTORY.ANCHOR]COMMON_FIELD_DEFINITIONS.ADDRESS.
BUSINESS_TYPE.
CONTACT_PERSON.
END RECORD.
SET DEFAULT DEVICE:[DIRECTORY.ANCHOR]INVENTORY
DEFINE FIELD PART DATATYPE IS TEXT SIZE IS 25 CHARACTERS.
DEFINE FIELD PART_NUMBER DATATYPE IS TEXT SIZE IS 10 CHARACTERS.
DEFINE RECORD INVENTORY_RECORD.
PART.
PART_NUMBER.
END RECORD.

REPLACE

REPLACE

Function

The REPLACE statement is used to replace source program text.

[pseudo-text-1]

is a text-matching argument that the compiler compares against text-words in the source text.

[pseudo-text-2]

is a replacement item that the compiler inserts into the source program.
Syntax Rules

1. A REPLACE statement can be inserted anywhere that a character-string can be used. This statement must be preceded by a separator period unless it is the first statement in a separately compiled program.

2. A REPLACE statement must be terminated by the separator period.

3. *pseudo-text-1* must contain at least one text-word.

4. *pseudo-text-2* can contain zero, one, or more text-words.

5. Character-strings within *pseudo-text-1* and *pseudo-text-2* can be continued.

6. *pseudo-text-1* must not consist entirely of a separator comma or a separator semicolon.

7. The word REPLACE is considered part of a comment-entry if it appears in the comment-entry or in the place where a comment-entry can appear.

General Rules

Format 1

1. Each matched occurrence of *pseudo-text-1* in the source program is replaced by the corresponding *pseudo-text-2*.

Format 2

2. Any text replacement currently in effect is discontinued.

Both Formats

3. A REPLACE statement remains in effect until the next occurrence of a REPLACE statement or until the end of a separately compiled program has been reached.

4. Any occurrence of a REPLACE statement in a source program is processed after all COPY statements in the source program have been processed.

5. *pseudo-text-2* must not contain a REPLACE statement.

6. The comparison operation starts with the leftmost source text word and the first text-matching argument. The compiler compares the entire text-matching argument to an equivalent number of consecutive source text-words.

7. A text-matching argument matches the source text only if the ordered sequence of text-words that forms the text-matching argument is equal, character for character, to the ordered sequence of source text-words.

   In the matching operation, the compiler treats each occurrence or combination of the following items in source text as a single space:
   - Separator comma
   - Separator semicolon
   - A sequence of one or more separator spaces

8. If no match occurs, the compiler repeats the comparison operation with each successive text-matching argument until a match is found or there are no more text-matching arguments.
9. If no match occurs after the compiler has compared all of the text-matching arguments, the next successive source text-word becomes the leftmost text-word, and the comparison resumes with the first occurrence of pseudo-text-1.

10. If a match occurs between a text-matching argument and the source program text, the compiler inserts the replacement text into the source program. The source text-word immediately following the rightmost replaced text-word becomes the leftmost text-word for the next cycle. The comparison cycle resumes with the first occurrence of pseudo-text-1.

11. The comparison cycles continue until the rightmost text-word in the source text that is within the scope of the REPLACE statement has been either:
   - Matched and replaced
   - Used as the leftmost source text-word in a comparison cycle

12. The rules for Reference Format determine the sequence of text-words in the source text and the text-matching arguments.

13. The compiler ignores comment lines and blank lines in the source program and in pseudo-text-1 for matching.

14. When the compiler inserts pseudo-text-2 in the source program, it inserts comment lines and blank lines in pseudo-text-2 without modification.

15. Debugging lines are permitted in pseudo-text-1 and pseudo-text-2. The compiler treats the comparison of debugging lines as if the conditional compilation character does not appear in the indicator area.

16. The compiler cannot determine the syntactic correctness of source text or the source program until all COPY and REPLACE statements have been processed.

17. Text words that are inserted as a result of a processed REPLACE statement are placed in the source program according to the rules for Reference Format.

18. When the compiler inserts text words of pseudo-text-2 into the source program, additional spaces may be introduced between text words where spaces already exist (including the assumed space between source lines).

19. If additional lines are added to the source program as a result of a REPLACE operation, the indicator area of the added lines contains the same character as the line on which the text being replaced begins (unless that line contains a hyphen, in which case the introduced line contains a space).

   If a literal within pseudo-text-2 cannot be contained on a single line without a continuation to another line in the resultant program and the literal is not being placed on a debugging line, additional continuation lines are introduced that contain the remainder of the literal. If replacement requires the continued literal to be continued on a debugging line, the program is in error.

Additional Reference

See Section 1.3: Source Reference Format.

Examples

In the following examples, uppercase words represent text-words that have been replaced.

1. REPLACE statement with multiple replacement items:

   8 working-storage section.
   9 replace ==alpha== by ==NUM-1==
   10 ==num== by ==ALPHA-1==.
   R 11 01  NUM-1  pic 9(10).
   R 12 01  ALPHA-1

458
Source Text Manipulation

2. **Multiple REPLACE statements:**

A given occurrence of the REPLACE statement is in effect from the point at which it is specified until the next occurrence of the REPLACE statement. The new REPLACE statement supersedes the text-matching established by the previous REPLACE statement.

```
    01 total           pic 9(4)v99.
    01 customer-rec.
    03 CLASS1      pic x(02).
    03 name        pic x(25).
    03 address.
    05 street  pic x(20).
    05 city    pic x(20).
    05 state   pic xx.
    05 zip     pic 9(5).
    03 orders occurs 6 times.
```

```
    procedure division.
    replace ==class== by ==CLASS1==.
    replace ==total== by ==ORDER-AMT==.
```

In the previous example, the word total on line 25 is not replaced because the REPLACE statement on line 24 reestablished the text-matching arguments.

3. **REPLACE OFF:**

Any text-matching currently in effect is turned off.

```
    01 a1              PIC 9(18).
    01 a2              PIC 9(18).
```

```
    procedure division.
    replace off.
    p0. add a1 to a2.
```

In the previous example, the word add on line 17 is not replaced because the REPLACE statement on line 16 turned off all text-matching arguments.

4. **COPY interaction:**

In the following example, library text is copied from the library file DATAFILE.LIB:

Contents of "DATAFILE.LIB":

```
  01 customer-rec.
  03 class       pic x(02).
  03 name        pic x(25).
  03 address.
      05 street  pic x(20).
      05 city    pic x(20).
      05 state   pic xx.
      05 zip     pic 9(5).
```
03 orders occurs 6 times.
  05 order-number  pic 9(6).
  05 order-date    pic 9(6).
  05 order-amt     pic 9(4)v99.

The text-matching specified by an active REPLACE statement occurs after COPY (and COPY REPLACING) processing is complete.

    7 working-storage section.
    8 replace ==class== by ==CLASS1==.
    9 copy datafile.
L 10 01 customer-rec.
L 11 03 CLASS1      pic x(02).
L 12 03 name        pic x(25).
L 13 03 address.
L 14   05 street  pic x(20).
L 15   05 city    pic x(20).
L 16   05 state   pic xx.
L 17   05 zip     pic 9(5).
L 18 03 orders occurs 6 times.
L 19   05 order-number  pic 9(6).
L 20   05 order-date    pic 9(6).
L 21   05 order-amt     pic 9(4)v99.
22 procedure division.
### Appendix A. VSI COBOL Reserved Words

<table>
<thead>
<tr>
<th>DIGITAL COBOL Reserved Words</th>
<th>ACCEPT</th>
<th>ACCESS</th>
<th>ADD</th>
<th>ADVANCING</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFTER</td>
<td>ALL</td>
<td>ALLOWING</td>
<td>ALPHABET</td>
<td></td>
</tr>
<tr>
<td>ALPHABETIC</td>
<td>ALPHABETIC–LOWER</td>
<td>ALPHABETIC–UPPER</td>
<td>ALPHANUMERIC</td>
<td></td>
</tr>
<tr>
<td>ALPHANUMERIC–EDITED</td>
<td>ALSO</td>
<td>ALTER</td>
<td>ALTERNATE</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>ANY</td>
<td>APPLY</td>
<td>ARE</td>
<td></td>
</tr>
<tr>
<td>AREA</td>
<td>AREAS</td>
<td>ASCENDING</td>
<td>ASSIGN</td>
<td></td>
</tr>
<tr>
<td>AT</td>
<td>AUTHOR</td>
<td>AUTO</td>
<td>AUTOMATIC</td>
<td></td>
</tr>
<tr>
<td>AUTOTERMINATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BACKGROUND-COLOR</td>
<td>BATCH</td>
<td>BEFORE</td>
<td>BEGINNING</td>
<td></td>
</tr>
<tr>
<td>BELL</td>
<td>BINARY</td>
<td>BIT</td>
<td>BITS</td>
<td></td>
</tr>
<tr>
<td>BLANK</td>
<td>BLINK</td>
<td>BLINKING</td>
<td>BLOCK</td>
<td></td>
</tr>
<tr>
<td>BOLD</td>
<td>BOOLEAN</td>
<td>BOTTOM</td>
<td>BY</td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>CANCEL</td>
<td>CD</td>
<td>CF</td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>CHARACTER</td>
<td>CHARACTERS</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>CLOCK-UNITS</td>
<td>CLOSE</td>
<td>COBOL</td>
<td>CODE</td>
<td></td>
</tr>
<tr>
<td>CODE-SET</td>
<td>COLLATING</td>
<td>COLUMN</td>
<td>COMMA</td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td>COMMON</td>
<td>COMMUNICATION</td>
<td>COMP</td>
<td></td>
</tr>
<tr>
<td>COMP-1</td>
<td>COMP-2</td>
<td>COMP-3</td>
<td>COMP-4</td>
<td></td>
</tr>
<tr>
<td>COMP-5</td>
<td>COMP-6</td>
<td>COMP-X</td>
<td>COMPUTATIONAL</td>
<td></td>
</tr>
<tr>
<td>COMPUTATIONAL-1</td>
<td>COMPUTATIONAL-2</td>
<td>COMPUTATIONAL-3</td>
<td>COMPUTATIONAL-4</td>
<td></td>
</tr>
<tr>
<td>COMPUTATIONAL-5</td>
<td>COMPUTATIONAL-6</td>
<td>COMPUTATIONAL-X</td>
<td>COMPUTE</td>
<td></td>
</tr>
<tr>
<td>CONCURRENT</td>
<td>CONFIGURATION</td>
<td>CONNECT</td>
<td>CONTAIN</td>
<td></td>
</tr>
<tr>
<td>CONTAINS</td>
<td>CONTENT</td>
<td>CONTINUE</td>
<td>CONTROL</td>
<td></td>
</tr>
<tr>
<td>CONTROLS</td>
<td>CONVERSION</td>
<td>CONVERTING</td>
<td>COPY</td>
<td></td>
</tr>
<tr>
<td>CORR</td>
<td>CORRESPONDING</td>
<td>COUNT</td>
<td>CRT</td>
<td></td>
</tr>
<tr>
<td>CURRENCY</td>
<td>CURRENT</td>
<td>CURSOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>DATE</td>
<td>DATE-COMPiled</td>
<td>DATE-WRITTEn</td>
<td></td>
</tr>
<tr>
<td>DAY</td>
<td>DAY-OF-WEEK</td>
<td>DB</td>
<td>DB-ACCESS-CONTROL-KEY</td>
<td></td>
</tr>
<tr>
<td>DB-CONDITION</td>
<td>DB-CURRENT-RECORD-ID</td>
<td>DB-CURRENT-RECORD-NAME</td>
<td>DB-EXCEPTION</td>
<td></td>
</tr>
<tr>
<td>DBKEY</td>
<td>DB-KEY</td>
<td>DB-RECORD-NAME</td>
<td>DB-SET-NAME</td>
<td></td>
</tr>
<tr>
<td>DB-STATUS</td>
<td>DB-UWA</td>
<td>DE</td>
<td>DEBUG-CONTENTS</td>
<td></td>
</tr>
<tr>
<td>DEBUG-ITEM</td>
<td>DEBUG-LENGTH</td>
<td>DEBUG-LINE</td>
<td>DEBUG-NAME</td>
<td></td>
</tr>
<tr>
<td>Digital COBOL Reserved Words</td>
<td>Debug-Numeric-Contents</td>
<td>Debug-Size</td>
<td>Debug-Start</td>
<td>Debug-Sub</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------</td>
<td>------------</td>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Debug-Sub-1</td>
<td>Debug-Sub-2</td>
<td>Debug-Sub-3</td>
<td>Debug-Sub-Item</td>
<td></td>
</tr>
<tr>
<td>Debug-Sub-N</td>
<td>Debug-Sub-Num</td>
<td>Debugging</td>
<td>Decimal-Point</td>
<td></td>
</tr>
<tr>
<td>Declaratives</td>
<td>Default</td>
<td>Delete</td>
<td>Delimited</td>
<td></td>
</tr>
<tr>
<td>Delimiter</td>
<td>Dependency</td>
<td>Depending</td>
<td>Descending</td>
<td></td>
</tr>
<tr>
<td>Descriptor</td>
<td>Destination</td>
<td>Detail</td>
<td>Dictionary</td>
<td></td>
</tr>
<tr>
<td>Disable</td>
<td>Disconnect</td>
<td>Display</td>
<td>Display-6</td>
<td></td>
</tr>
<tr>
<td>Display-7</td>
<td>Display-9</td>
<td>Divide</td>
<td>Division</td>
<td></td>
</tr>
<tr>
<td>Does</td>
<td>Down</td>
<td>Duplicate(s)</td>
<td>Dynamic</td>
<td></td>
</tr>
<tr>
<td>Echo</td>
<td>Editing</td>
<td>EGI</td>
<td>Else</td>
<td></td>
</tr>
<tr>
<td>EMI</td>
<td>Empty</td>
<td>Enable</td>
<td>End</td>
<td></td>
</tr>
<tr>
<td>End-Accept</td>
<td>End-Add</td>
<td>End-Call</td>
<td>End-Commit</td>
<td></td>
</tr>
<tr>
<td>End-Compute</td>
<td>End-Connect</td>
<td>End-Delete</td>
<td>End-Disconnect</td>
<td></td>
</tr>
<tr>
<td>End-Divide</td>
<td>End-Erase</td>
<td>End-Evaluate</td>
<td>End-Fetch</td>
<td></td>
</tr>
<tr>
<td>End-Find</td>
<td>End-Finish</td>
<td>End-Free</td>
<td>End-Get</td>
<td></td>
</tr>
<tr>
<td>End-If</td>
<td>End-Keep</td>
<td>End-Multiply</td>
<td>End-Multiply</td>
<td></td>
</tr>
<tr>
<td>End-Of-Page</td>
<td>End-Perform</td>
<td>End-Read</td>
<td>End-Ready</td>
<td></td>
</tr>
<tr>
<td>End-Receive</td>
<td>End-Reconnect</td>
<td>End-Return</td>
<td>End-Rewrite</td>
<td></td>
</tr>
<tr>
<td>End-Rollback</td>
<td>End-Search</td>
<td>End-Start</td>
<td>End-Store</td>
<td></td>
</tr>
<tr>
<td>End-String</td>
<td>End-Subtract</td>
<td>End-Unstring</td>
<td>End-Write</td>
<td></td>
</tr>
<tr>
<td>Ending</td>
<td>Enter</td>
<td>Environment</td>
<td>Eol</td>
<td></td>
</tr>
<tr>
<td>Eop</td>
<td>Eos</td>
<td>Equal</td>
<td>Equals</td>
<td></td>
</tr>
<tr>
<td>Erase</td>
<td>Error</td>
<td>ESI</td>
<td>Evaluate</td>
<td></td>
</tr>
<tr>
<td>Every</td>
<td>Exceeds</td>
<td>Exception</td>
<td>Exclusive</td>
<td></td>
</tr>
<tr>
<td>Exit</td>
<td>Exor</td>
<td>Extend</td>
<td>External</td>
<td></td>
</tr>
<tr>
<td>Failure</td>
<td>False</td>
<td>Fd</td>
<td>Fetch</td>
<td></td>
</tr>
<tr>
<td>File</td>
<td>File-Control</td>
<td>Filler</td>
<td>Final</td>
<td></td>
</tr>
<tr>
<td>Find</td>
<td>Finish</td>
<td>First</td>
<td>Footing</td>
<td></td>
</tr>
<tr>
<td>For</td>
<td>Foreground-Color</td>
<td>Free</td>
<td>From</td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>Function</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate</td>
<td>Get</td>
<td>Giving</td>
<td>Global</td>
<td></td>
</tr>
<tr>
<td>Go</td>
<td>Greater</td>
<td>Group</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heading</td>
<td>Highlight</td>
<td>High-Value(s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ident</td>
<td>Identification</td>
<td>If</td>
<td>In</td>
<td></td>
</tr>
<tr>
<td>Including</td>
<td>Index</td>
<td>Indexed</td>
<td>Indicate</td>
<td></td>
</tr>
<tr>
<td>Reserved Words</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INITIAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INITIALIZE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INITIATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPUT-OUTPUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSPECT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSTALLATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INVALID</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I-O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I-O-CONTROL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUST</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUSTIFIED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEEP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LABEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAST</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEADING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEFT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LENGTH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LESS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIMITS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINAGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINAGE-COUNTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINE-COUNTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINKAGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCALLY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCK-HOLDING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOWLIGHT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOW-VALUE(S)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MANUAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATCHES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEMBER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEMBERSHIP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEMORY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MERGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MESSAGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODIFY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODULES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULTIPLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULTIPLY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NATIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEGATIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEXT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NON-NULL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMBER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMERIC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMERIC-EDITED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBJECT-COMPUTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCCURS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OFFSET</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OMITTED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONLY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPTIONAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORDER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORGANIZATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OTHER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OTHERS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUTPUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVERFLOW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OWNER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PACKED-DECIMAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PADDING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGE-COUNTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERFORM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PICTURE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLUS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POINTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POSITION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POSITIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRINTING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRIOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROCEDURE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROCEDURES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROCEED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROGRAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROGRAM-ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROTECTED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PURGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PREVIOUS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUEUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTE(S)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RANDOM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READERS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REALM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REALMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECEIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECONNECT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECORD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECORD-NAME</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECORDS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved Words</td>
<td>DIGITAL COBOL Reserved Words</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REDEFINES</td>
<td>REEL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REFERENCES</td>
<td>REGARDLESS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REMAINDER</td>
<td>REMOVAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPLACING</td>
<td>REPORT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQUIRED</td>
<td>RERUN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETAINING</td>
<td>RETRIEVAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REVERSED</td>
<td>REVERSE-VIDEO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>RH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMS-CURRENT-STS</td>
<td>RHS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAME</td>
<td>SCREEN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECTION</td>
<td>SECURE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEGMENT-LIMIT</td>
<td>SELECT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEPARATE</td>
<td>SEQUENCE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td>SETS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SORT</td>
<td>SORT-MERGE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPACE</td>
<td>SPACES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STANDARD-1</td>
<td>STANDARD-2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STOP</td>
<td>STORE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB-QUEUE-2</td>
<td>SUB-QUEUE-3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUCCESS</td>
<td>SUM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYNC</td>
<td>SYNCHRONIZED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE</td>
<td>TALLYING</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERMINAL</td>
<td>TERMINATE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>THAN</td>
<td>THEN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>TIMES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRAILING</td>
<td>TRUE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNDERLINE</td>
<td>UNDERLINED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNLOCK</td>
<td>UNSTRING</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPDATE</td>
<td>UPDATING</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USAGE-MODE</td>
<td>USE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VALUE</td>
<td>VALUES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAIT</td>
<td>WHEN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WITHIN</td>
<td>WORDS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIGITAL COBOL Reserved Words</th>
<th>DIGITAL COBOL Reserved Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFERENCES</td>
<td>RELATIVE</td>
</tr>
<tr>
<td>REMAINDER</td>
<td>NAMES</td>
</tr>
<tr>
<td>REPLACING</td>
<td>REPORTING</td>
</tr>
<tr>
<td>REQUIRED</td>
<td>RESERVE</td>
</tr>
<tr>
<td>RETAINING</td>
<td>RETURN</td>
</tr>
<tr>
<td>REVERSED</td>
<td>REWRITE</td>
</tr>
<tr>
<td>RF</td>
<td>RMS-CURRENT-FILENAME</td>
</tr>
<tr>
<td>RMS-CURRENT-STV</td>
<td>RMS-CURRENT-STV</td>
</tr>
<tr>
<td>RMS-FILENAME</td>
<td>RMS-FILENAME</td>
</tr>
<tr>
<td>SAME</td>
<td>SD</td>
</tr>
<tr>
<td>SECTION</td>
<td>SECURITY</td>
</tr>
<tr>
<td>SEGMENT-LIMIT</td>
<td>SEND</td>
</tr>
<tr>
<td>SEPARATE</td>
<td>SEQUENCE-LENGTH</td>
</tr>
<tr>
<td>SET</td>
<td>SIGN</td>
</tr>
<tr>
<td>SORT</td>
<td>SOURCE</td>
</tr>
<tr>
<td>SPACE</td>
<td>SPECIAL-NAME</td>
</tr>
<tr>
<td>STANDARD-1</td>
<td>START</td>
</tr>
<tr>
<td>STOP</td>
<td>STRING</td>
</tr>
<tr>
<td>SUB-QUEUE-2</td>
<td>SUB-QUEUE-3</td>
</tr>
<tr>
<td>SUCCESS</td>
<td>SUM</td>
</tr>
<tr>
<td>SYNC</td>
<td>SYNCHRONIZED</td>
</tr>
<tr>
<td>TABLE</td>
<td>TAPE</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>TEST</td>
</tr>
<tr>
<td>THAN</td>
<td>THROUGH</td>
</tr>
<tr>
<td>TIME</td>
<td>TO</td>
</tr>
<tr>
<td>TRAILING</td>
<td>TRUE</td>
</tr>
<tr>
<td>UNDERLINE</td>
<td>UNDERLINED</td>
</tr>
<tr>
<td>UNLOCK</td>
<td>UNSTRING</td>
</tr>
<tr>
<td>UPDATE</td>
<td>UPDATING</td>
</tr>
<tr>
<td>USAGE-MODE</td>
<td>USE</td>
</tr>
<tr>
<td>VALUE</td>
<td>VALUES</td>
</tr>
<tr>
<td>WAIT</td>
<td>WHEN</td>
</tr>
<tr>
<td>WITHIN</td>
<td>WORDS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIGITAL COBOL Reserved Words</th>
<th>DIGITAL COBOL Reserved Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFERENCES</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>REMAINDER</td>
<td>RETAINING</td>
</tr>
<tr>
<td>REPLACING</td>
<td>RECOVER</td>
</tr>
<tr>
<td>REQUIRED</td>
<td>RESERVE</td>
</tr>
<tr>
<td>RETAINING</td>
<td>RETURN</td>
</tr>
<tr>
<td>REVERSED</td>
<td>REPLACE</td>
</tr>
<tr>
<td>RF</td>
<td>RMS-CURRENT-FILENAME</td>
</tr>
<tr>
<td>RMS-CURRENT-STV</td>
<td>RMS-CURRENT-STV</td>
</tr>
<tr>
<td>RMS-FILENAME</td>
<td>RMS-CURRENT</td>
</tr>
<tr>
<td>SAME</td>
<td>SD</td>
</tr>
<tr>
<td>SECTION</td>
<td>SECURITY</td>
</tr>
<tr>
<td>SEGMENT-LIMIT</td>
<td>SEND</td>
</tr>
<tr>
<td>SEPARATE</td>
<td>SEQUENCE-LENGTH</td>
</tr>
<tr>
<td>SET</td>
<td>SIGN</td>
</tr>
<tr>
<td>SORT</td>
<td>SOURCE</td>
</tr>
<tr>
<td>SPACE</td>
<td>SPECIAL-NAME</td>
</tr>
<tr>
<td>STANDARD-1</td>
<td>START</td>
</tr>
<tr>
<td>STOP</td>
<td>STRING</td>
</tr>
<tr>
<td>SUB-QUEUE-2</td>
<td>SUB-QUEUE-3</td>
</tr>
<tr>
<td>SUCCESS</td>
<td>SUM</td>
</tr>
<tr>
<td>SYNC</td>
<td>SYNCHRONIZED</td>
</tr>
<tr>
<td>TABLE</td>
<td>TAPE</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>TEST</td>
</tr>
<tr>
<td>THAN</td>
<td>THROUGH</td>
</tr>
<tr>
<td>TIME</td>
<td>TO</td>
</tr>
<tr>
<td>TRAILING</td>
<td>TRUE</td>
</tr>
<tr>
<td>UNDERLINE</td>
<td>UNDERLINED</td>
</tr>
<tr>
<td>UNLOCK</td>
<td>UNSTRING</td>
</tr>
<tr>
<td>UPDATE</td>
<td>UPDATING</td>
</tr>
<tr>
<td>USAGE-MODE</td>
<td>USE</td>
</tr>
<tr>
<td>VALUE</td>
<td>VALUES</td>
</tr>
<tr>
<td>WAIT</td>
<td>WHEN</td>
</tr>
<tr>
<td>WITHIN</td>
<td>WORDS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIGITAL COBOL Reserved Words</th>
<th>DIGITAL COBOL Reserved Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFERENCES</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>REMAINDER</td>
<td>RETAINING</td>
</tr>
<tr>
<td>REPLACING</td>
<td>RECOVER</td>
</tr>
<tr>
<td>REQUIRED</td>
<td>RESERVE</td>
</tr>
<tr>
<td>RETAINING</td>
<td>RETURN</td>
</tr>
<tr>
<td>REVERSED</td>
<td>REPLACE</td>
</tr>
<tr>
<td>RF</td>
<td>RMS-CURRENT-FILENAME</td>
</tr>
<tr>
<td>RMS-CURRENT-STV</td>
<td>RMS-CURRENT-STV</td>
</tr>
<tr>
<td>RMS-FILENAME</td>
<td>RMS-CURRENT</td>
</tr>
<tr>
<td>SAME</td>
<td>SD</td>
</tr>
<tr>
<td>SECTION</td>
<td>SECURITY</td>
</tr>
<tr>
<td>SEGMENT-LIMIT</td>
<td>SEND</td>
</tr>
<tr>
<td>SEPARATE</td>
<td>SEQUENCE-LENGTH</td>
</tr>
<tr>
<td>SET</td>
<td>SIGN</td>
</tr>
<tr>
<td>SORT</td>
<td>SOURCE</td>
</tr>
<tr>
<td>SPACE</td>
<td>SPECIAL-NAME</td>
</tr>
<tr>
<td>STANDARD-1</td>
<td>START</td>
</tr>
<tr>
<td>STOP</td>
<td>STRING</td>
</tr>
<tr>
<td>SUB-QUEUE-2</td>
<td>SUB-QUEUE-3</td>
</tr>
<tr>
<td>SUCCESS</td>
<td>SUM</td>
</tr>
<tr>
<td>SYNC</td>
<td>SYNCHRONIZED</td>
</tr>
<tr>
<td>TABLE</td>
<td>TAPE</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>TEST</td>
</tr>
<tr>
<td>THAN</td>
<td>THROUGH</td>
</tr>
<tr>
<td>TIME</td>
<td>TO</td>
</tr>
<tr>
<td>TRAILING</td>
<td>TRUE</td>
</tr>
<tr>
<td>UNDERLINE</td>
<td>UNDERLINED</td>
</tr>
<tr>
<td>UNLOCK</td>
<td>UNSTRING</td>
</tr>
<tr>
<td>UPDATE</td>
<td>UPDATING</td>
</tr>
<tr>
<td>USAGE-MODE</td>
<td>USE</td>
</tr>
<tr>
<td>VALUE</td>
<td>VALUES</td>
</tr>
<tr>
<td>WAIT</td>
<td>WHEN</td>
</tr>
<tr>
<td>WITHIN</td>
<td>WORDS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIGITAL COBOL Reserved Words</th>
<th>DIGITAL COBOL Reserved Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFERENCES</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>REMAINDER</td>
<td>RETAINING</td>
</tr>
<tr>
<td>REPLACING</td>
<td>RECOVER</td>
</tr>
<tr>
<td>REQUIRED</td>
<td>RESERVE</td>
</tr>
<tr>
<td>RETAINING</td>
<td>RETURN</td>
</tr>
<tr>
<td>REVERSED</td>
<td>REPLACE</td>
</tr>
<tr>
<td>RF</td>
<td>RMS-CURRENT-FILENAME</td>
</tr>
<tr>
<td>RMS-CURRENT-STV</td>
<td>RMS-CURRENT-STV</td>
</tr>
<tr>
<td>RMS-FILENAME</td>
<td>RMS-CURRENT</td>
</tr>
<tr>
<td>SAME</td>
<td>SD</td>
</tr>
<tr>
<td>SECTION</td>
<td>SECURITY</td>
</tr>
<tr>
<td>SEGMENT-LIMIT</td>
<td>SEND</td>
</tr>
<tr>
<td>SEPARATE</td>
<td>SEQUENCE-LENGTH</td>
</tr>
<tr>
<td>SET</td>
<td>SIGN</td>
</tr>
<tr>
<td>SORT</td>
<td>SOURCE</td>
</tr>
<tr>
<td>SPACE</td>
<td>SPECIAL-NAME</td>
</tr>
<tr>
<td>STANDARD-1</td>
<td>START</td>
</tr>
<tr>
<td>STOP</td>
<td>STRING</td>
</tr>
<tr>
<td>SUB-QUEUE-2</td>
<td>SUB-QUEUE-3</td>
</tr>
<tr>
<td>SUCCESS</td>
<td>SUM</td>
</tr>
<tr>
<td>SYNC</td>
<td>SYNCHRONIZED</td>
</tr>
<tr>
<td>TABLE</td>
<td>TAPE</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>TEST</td>
</tr>
<tr>
<td>THAN</td>
<td>THROUGH</td>
</tr>
<tr>
<td>TIME</td>
<td>TO</td>
</tr>
<tr>
<td>TRAILING</td>
<td>TRUE</td>
</tr>
<tr>
<td>UNDERLINE</td>
<td>UNDERLINED</td>
</tr>
<tr>
<td>UNLOCK</td>
<td>UNSTRING</td>
</tr>
<tr>
<td>UPDATE</td>
<td>UPDATING</td>
</tr>
<tr>
<td>USAGE-MODE</td>
<td>USE</td>
</tr>
<tr>
<td>VALUE</td>
<td>VALUES</td>
</tr>
<tr>
<td>WAIT</td>
<td>WHEN</td>
</tr>
<tr>
<td>WITHIN</td>
<td>WORDS</td>
</tr>
</tbody>
</table>
## DIGITAL COBOL Reserved Words

<table>
<thead>
<tr>
<th>Reserved Words</th>
<th>WRITERS</th>
<th>ZEROES</th>
<th>ZEROS</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
</tr>
<tr>
<td>**</td>
<td>&gt;</td>
<td>&lt;</td>
<td>=</td>
</tr>
<tr>
<td>&gt; =</td>
<td>&lt; =</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The reserved words listed in this appendix are both the default reserved words and the words that on Alpha and I64 systems are reserved only if activated by the COBOL command-line qualifier / RESERVED_WORDS=FOREIGN_EXTENSIONS or /RESERVED_WORDS=200X.

The XOPEN reserved words, which on Alpha and I64 systems are reserved by default, can be deactivated by the /RESERVED_WORDS=NOXOPEN qualifier.

These three categories of Alpha- and I64-only reserved words, which are activated or deactivated by command-line qualifiers, are marked in this appendix as follows:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[FOREIGN]</td>
<td>Reserved only if activated by / RESERVED_WORDS=FOREIGN_EXTENSIONS</td>
</tr>
<tr>
<td>[200X]</td>
<td>Reserved only if activated by / RESERVED_WORDS=200X</td>
</tr>
<tr>
<td>[XOPEN]</td>
<td>Reserved by default, but not reserved if deactivated by /RESERVED_WORDS=NOXOPEN</td>
</tr>
</tbody>
</table>

### Reserved Words

- ACCEPT
- ACCESS
- ADD
- ADDRESS [FOREIGN] (Alpha, I64)
- ADVANCING
- AFTER
- ALL
- ALLOWING
- ALPHABET
- ALPHABETIC
- ALPHABETIC–LOWER
- ALPHABETIC–UPPER
- ALPHANUMERIC
- ALPHANUMERIC–EDITED
- ALSO
- ALTER
- ALTERNATE
- AND
- ANY
- APPLY
- ARE
- AREA
- AREAS
- ASCENDING
- ASSIGN
- AT
AUTHOR
AUTO [XOPEN] (Alpha, I64)
AUTOMATIC
AUTOTERMINATE
BACKGROUND-COLOR [XOPEN] (Alpha, I64)
BATCH
BEFORE
BEGINNING
BELL [XOPEN] (Alpha, I64)
BINARY
BINARY-CHAR [200X] (Alpha, I64)
BINARY-DOUBLE [200X] (Alpha, I64)
BINARY-LONG [200X] (Alpha, I64)
BINARY-SHORT [200X] (Alpha, I64)
BIT
BITS
BLANK
BLINK [XOPEN] (Alpha, I64)
BLINKING
BLOCK
BOLD
BOOLEAN
BOTTOM
BY
CALL
CANCEL
CD
CF
CH
CHANGED [FOREIGN] (Alpha, I64)
CHARACTER
CHARACTERS
CLASS
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COL [200X] (Alpha, I64)
COLLATING
COLUMN
COMMA
COMMIT
COMMON
COMMUNICATION
COMP
COMP-1
COMP-2
COMP-3
COMP-4
COMP-5
COMP-6
COMP-X
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTATIONAL-4
Appendix A. VSI COBOL
Reserved Words

COMPUTATIONAL-5
COMPUTATIONAL-6
COMPUTATIONAL-X
COMPUTE
CONCURRENT
CONFIGURATION
CONNECT
CONTAIN
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROLS
CONVERSION
CONVERTING
COPY
CORE-INDEX [FOREIGN] (Alpha, I64)
CORR
CORRESPONDING
COUNT
CRT
CURRENCY
CURRENT
CURSOR
DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DAY-OF-WEEK
DB
DB-ACCESS-CONTROL-KEY
DB-CONDITION
DB-CURRENT-RECORD-ID
DB-CURRENT-RECORD-NAME
DB-EXCEPTION
DB-KEY
DB-RECORD-NAME
DB-SET-NAME
DB-STATUS
DB-UWA
DBCS [FOREIGN] (Alpha, I64)
DBKEY
DE
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LENGTH
DEBUG-LINE
DEBUG-NAME
DEBUG-NUMERIC-CONTENTS
DEBUG-SIZE
DEBUG-START
DEBUG-SUB
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUG-SUB-ITEM
DEBUG-SUB-N
Appendix A. VSI COBOL
Reserved Words

DEBUG-SUB-NUM
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DEFAULT
DELETE
DELIMITED
DELIMITER
DEPENDENCY
DEPENDING
DESCENDING
_DESCRIPTOR
DESTINATION
DETAIL
DICTIONARY
DISABLE
DISCONNECT
_DISP [FOREIGN] (Alpha, I64)
DISPLAY
_DISPLAY-1 [FOREIGN] (Alpha, I64)
DISPLAY-6
DISPLAY-7
DISPLAY-9
DIVIDE
DIVISION
DOES
DOWN
DUPPLICATE
DUPlicATES
ECHO
EDITING
EGI
_EJECT [FOREIGN] (Alpha, I64)
ELSE
EMI
EMPTY
ENABLE
END
END-ACCEPT
END-ADD
END-CALL
END-COMMIT
END-COMPUTE
END-CONNECT
END-DELETE
END-DISCONNECT
END-DIVIDE
END-ERASE
END-EVALUATE
END-FETCH
END-FIND
END-FINISH
END-FREE
END-GET
END-IF
END-KEEP
END-MODIFY
END-MULTIPLY
Appendix A. VSI COBOL
Reserved Words

END-OF-PAGE
END-PERFORM
END-READ
END-READY
END-RECEIVE
END-RECONNECT
END-RETURN
END-REWRITE
END-ROLLBACK
END-SEARCH
END-START
END-STORE
END-STRING
END-SUBTRACT
END-UNSTRING
END-WRITE
ENDING
ENTER
ENTRY [FOREIGN] (Alpha, I64)
ENVIRONMENT
EOL [XOPEN] (Alpha, I64)
EOP
EOS [XOPEN] (Alpha, I64)
EQUAL
EQUALS
ERASE [XOPEN] (Alpha, I64)
ERROR
ESI
EVALUATE
EVERY
EXAMINE [FOREIGN] (Alpha, I64)
EXCEEDS
EXCEPTION
EXCLUSIVE
EXHIBIT [FOREIGN] (Alpha, I64)
EXIT
EXOR
EXTEND
EXTERNAL
FAILURE
FALSE
FD
FETCH
FILE
FILE-CONTROL
FILLER
FINAL
FIND
FINISH
FIRST
FLOAT-EXTENDED [200X] (Alpha, I64)
FLOAT-LONG [200X] (Alpha, I64)
FLOAT-SHORT [200X] (Alpha, I64)
FOOTING
FOR
FOREGROUND-COLOR [XOPEN] (Alpha, I64)
FREE
FROM
FULL [XOPEN] (Alpha, I64)
FUNCTION
GENERATE
GET
GIVING
GLOBAL
GO
GOBACK [FOREIGN] (Alpha, I64)
GREATER
GROUP
HEADING
HIGH-VALUE
HIGH-VALUES
HIGHLIGHT [XOPEN] (Alpha, I64)
I-O
I-O-CONTROL
ID [FOREIGN] (Alpha, I64)
IDENT
IDENTIFICATION
IF
IN
INCLUDING
INDEX
INDEXED
INDICATE
INITIAL
INITIALIZE
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS
JUST
JUSTIFIED
KANJI [FOREIGN] (Alpha, I64)
KEEP
KEY
LABEL
LAST
LD
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCALLY
LOCK
LOCK-HOLDING
Appendix A. VSI COBOL
Reserved Words

LOW-VALUE
LOW-VALUES
LOWLIGHT \[XOPEN\] (Alpha, I64)
MANUAL
MATCH
MATCHES
MEMBER
MEMBERSHIP
MEMORY
MERGE
MESSAGE
MODE
MODIFY
MODULES
MOVE
MULTIPLE
MULTIPLY
NAMED \[FOREIGN\] (Alpha, I64)
NATIVE
NEGATIVE
NEXT
NO
NON-NUL
NOT
NOTE \[FOREIGN\] (Alpha, I64)
NULL
NUMBER
NUMERIC
NUMERIC-EDITED
OBJECT-COMPUTER
OCCURS
OF
OFF
OFFSET
OMITTED
ON
ONLY
OPEN
OPTIONAL
OPTIONS \[200X\] (Alpha, I64)
OR
ORDER
OTHERWISE \[FOREIGN\] (Alpha, I64)
PACKED-DECIMAL
PADDING
PAGE
PAGE-COUNTER
PASSWORD \[FOREIGN\] (Alpha, I64)
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIONING \[FOREIGN\] (Alpha, I64)
POSITIVE
PREVIOUS
PRINTING
PRIOR
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID
PROTECTED
PURGE
QUEUE
QUOTE
QUOTES
RANDOM
RD
READ
READERS
READY
REALM
REALMS
RECEIVE
RECONNECT
RECORD
RECORD-NAME
RECORD-OVERFLOW [FOREIGN] (Alpha, I64)
RECORDING [FOREIGN] (Alpha, I64)
RECORDS
REDEFINES
REEL
REFERENCE
REFERENCE-MODIFIER
REFERENCES
REGARDLESS
RELATIVE
RELEASE
RELOAD [FOREIGN] (Alpha, I64)
REMAINDER
REMARKS [FOREIGN] (Alpha, I64)
REMOVAL
RENAMES
REORG-CRITERIA [FOREIGN] (Alpha, I64)
REPLACE
REPLACING
REPORT
REPORTING
REPORTS
REQUIRED [XOPEN] (Alpha, I64)
RERUN
RESERVE
RESET
RETAINING
RETRIEVAL
RETURN
RETURN-CODE [XOPEN] (Alpha, I64)
RETURNING [FOREIGN] (Alpha, I64)
REVERSE-VIDEO [XOPEN] (Alpha, I64)
REVERSED
REWIND
Appendix A. VSI COBOL
Reserved Words

REWRITE
RF
RH
RIGHT
RMS-CURRENT-FILENAME
RMS-CURRENT-STS
RMS-CURRENT-STV
RMS-FILENAME
RMS-STS
RMS-STV
ROLLBACK
ROUNDED
RUN
SAME
SCREEN [XOPEN] (Alpha, I64)
SD
SEARCH
SECTION
SECURE [XOPEN] (Alpha, I64)
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENCE-NUMBER
SEQUENTIAL
SERVICE [FOREIGN] (Alpha, I64)
SET
SETS
SIGN
SIGNED [200X] (Alpha, I64)
SIZE
SKIP1 [FOREIGN] (Alpha, I64)
SKIP2 [FOREIGN] (Alpha, I64)
SKIP3 [FOREIGN] (Alpha, I64)
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
STANDARD-2
START
STATUS
STOP
STORE
STREAM
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUB-SCHEMA
SUBTRACT
SUCCESS
SUM
SUPPRESS
SYMBOL [200X] (Alpha, I64)
SYMBOLIC
SYNC
SYNCHRONIZED
TABLE
TALLYING
TAPE
TENANT
TERMINAL
TERMINATE
TEST
TEXT
THAN
THEN
THROUGH
THRU
TIME
TIMES
TO
TOP
TRACE [FOREIGN] (Alpha, I64)
TRAILING
TRANSFORM [FOREIGN] (Alpha, I64)
TRUE
TYPE
UNDERLINE [XOPEN] (Alpha, I64)
UNDERLINED
UNEQUAL
UNIT
UNLOCK
UNSIGNED [200X] (Alpha, I64)
UNSTRING
UNTIL
UP
UPDATE
UPDATERS
UPON
USAGE
USAGE-MODE
USE
USING
VALUE
VALUES
VARYING
VFU-CHANNEL
WAIT
WHEN
WHERE
WITH
WITHIN
WORDS
WORKING-STORAGE
WRITE
WRITERS
ZERO
ZEROES
ZEROS
+
-
*
/
**
>
< =
> =
< =
# Appendix B. Character Sets

<table>
<thead>
<tr>
<th>Position</th>
<th>Character</th>
<th>ASCII</th>
<th>EBCDIC</th>
<th>NATIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>NUL</td>
<td>000</td>
<td>00</td>
<td>000</td>
</tr>
<tr>
<td>002</td>
<td>SOH</td>
<td>001</td>
<td>01</td>
<td>001</td>
</tr>
<tr>
<td>003</td>
<td>STX</td>
<td>002</td>
<td>02</td>
<td>002</td>
</tr>
<tr>
<td>004</td>
<td>ETX</td>
<td>003</td>
<td>03</td>
<td>003</td>
</tr>
<tr>
<td>005</td>
<td>EOT</td>
<td>004</td>
<td>04</td>
<td>055</td>
</tr>
<tr>
<td>006</td>
<td>ENQ</td>
<td>005</td>
<td>05</td>
<td>045</td>
</tr>
<tr>
<td>007</td>
<td>ACK</td>
<td>006</td>
<td>06</td>
<td>046</td>
</tr>
<tr>
<td>008</td>
<td>BEL</td>
<td>007</td>
<td>07</td>
<td>047</td>
</tr>
<tr>
<td>009</td>
<td>BS</td>
<td>008</td>
<td>08</td>
<td>022</td>
</tr>
<tr>
<td>010</td>
<td>HT</td>
<td>009</td>
<td>09</td>
<td>005</td>
</tr>
<tr>
<td>011</td>
<td>LF</td>
<td>010</td>
<td>0A</td>
<td>037</td>
</tr>
<tr>
<td>012</td>
<td>VT</td>
<td>011</td>
<td>0B</td>
<td>011</td>
</tr>
<tr>
<td>013</td>
<td>FF</td>
<td>012</td>
<td>0C</td>
<td>012</td>
</tr>
<tr>
<td>014</td>
<td>CR</td>
<td>013</td>
<td>0D</td>
<td>013</td>
</tr>
<tr>
<td>015</td>
<td>SO</td>
<td>014</td>
<td>0E</td>
<td>014</td>
</tr>
<tr>
<td>016</td>
<td>SI</td>
<td>015</td>
<td>0F</td>
<td>015</td>
</tr>
<tr>
<td>017</td>
<td>DLE</td>
<td>016</td>
<td>10</td>
<td>016</td>
</tr>
<tr>
<td>018</td>
<td>DC1</td>
<td>017</td>
<td>11</td>
<td>017</td>
</tr>
<tr>
<td>019</td>
<td>DC2</td>
<td>018</td>
<td>12</td>
<td>018</td>
</tr>
<tr>
<td>020</td>
<td>DC3</td>
<td>019</td>
<td>13</td>
<td>019</td>
</tr>
<tr>
<td>021</td>
<td>DC4</td>
<td>020</td>
<td>14</td>
<td>060</td>
</tr>
<tr>
<td>022</td>
<td>NAK</td>
<td>021</td>
<td>15</td>
<td>061</td>
</tr>
<tr>
<td>023</td>
<td>SYN</td>
<td>022</td>
<td>16</td>
<td>050</td>
</tr>
<tr>
<td>024</td>
<td>ETB</td>
<td>023</td>
<td>17</td>
<td>038</td>
</tr>
<tr>
<td>025</td>
<td>CAN</td>
<td>024</td>
<td>18</td>
<td>024</td>
</tr>
<tr>
<td>026</td>
<td>EM</td>
<td>025</td>
<td>19</td>
<td>025</td>
</tr>
<tr>
<td>027</td>
<td>SUB</td>
<td>026</td>
<td>1A</td>
<td>063</td>
</tr>
<tr>
<td>028</td>
<td>ESC</td>
<td>027</td>
<td>1B</td>
<td>039</td>
</tr>
<tr>
<td>029</td>
<td>FS</td>
<td>028</td>
<td>1C</td>
<td>028</td>
</tr>
<tr>
<td>030</td>
<td>GS</td>
<td>029</td>
<td>1D</td>
<td>029</td>
</tr>
<tr>
<td>031</td>
<td>RS</td>
<td>030</td>
<td>1E</td>
<td>030</td>
</tr>
<tr>
<td>032</td>
<td>US</td>
<td>031</td>
<td>1F</td>
<td>031</td>
</tr>
<tr>
<td>033</td>
<td>space</td>
<td>032</td>
<td>20</td>
<td>064</td>
</tr>
<tr>
<td>034</td>
<td>!</td>
<td>033</td>
<td>21</td>
<td>090</td>
</tr>
</tbody>
</table>

477
### Appendix B. Character Sets

<table>
<thead>
<tr>
<th>Position</th>
<th>Character</th>
<th>ASCII Dec</th>
<th>ASCII Hex</th>
<th>EBCDIC Dec</th>
<th>EBCDIC Hex</th>
<th>NATIVE Dec</th>
<th>NATIVE Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>035</td>
<td>&quot;</td>
<td>034</td>
<td>22</td>
<td>127</td>
<td>7F</td>
<td>034</td>
<td>22</td>
</tr>
<tr>
<td>036</td>
<td>#</td>
<td>035</td>
<td>23</td>
<td>123</td>
<td>7B</td>
<td>035</td>
<td>23</td>
</tr>
<tr>
<td>037</td>
<td>$</td>
<td>036</td>
<td>24</td>
<td>091</td>
<td>5B</td>
<td>036</td>
<td>24</td>
</tr>
<tr>
<td>038</td>
<td>%</td>
<td>037</td>
<td>25</td>
<td>108</td>
<td>6C</td>
<td>037</td>
<td>25</td>
</tr>
<tr>
<td>039</td>
<td>&amp;</td>
<td>038</td>
<td>26</td>
<td>080</td>
<td>50</td>
<td>038</td>
<td>26</td>
</tr>
<tr>
<td>040</td>
<td>`</td>
<td>039</td>
<td>27</td>
<td>125</td>
<td>7D</td>
<td>039</td>
<td>27</td>
</tr>
<tr>
<td>041</td>
<td>(</td>
<td>040</td>
<td>28</td>
<td>077</td>
<td>4D</td>
<td>040</td>
<td>28</td>
</tr>
<tr>
<td>042</td>
<td>)</td>
<td>041</td>
<td>29</td>
<td>093</td>
<td>5D</td>
<td>041</td>
<td>29</td>
</tr>
<tr>
<td>043</td>
<td>*</td>
<td>042</td>
<td>2A</td>
<td>092</td>
<td>5C</td>
<td>042</td>
<td>2A</td>
</tr>
<tr>
<td>044</td>
<td>+</td>
<td>043</td>
<td>2B</td>
<td>078</td>
<td>4E</td>
<td>043</td>
<td>2B</td>
</tr>
<tr>
<td>045</td>
<td>,</td>
<td>044</td>
<td>2C</td>
<td>107</td>
<td>6B</td>
<td>044</td>
<td>2C</td>
</tr>
<tr>
<td>046</td>
<td>-</td>
<td>045</td>
<td>2D</td>
<td>096</td>
<td>60</td>
<td>045</td>
<td>2D</td>
</tr>
<tr>
<td>047</td>
<td>.</td>
<td>046</td>
<td>2E</td>
<td>075</td>
<td>4B</td>
<td>046</td>
<td>2E</td>
</tr>
<tr>
<td>048</td>
<td>/</td>
<td>047</td>
<td>2F</td>
<td>097</td>
<td>61</td>
<td>047</td>
<td>2F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Position</th>
<th>Character</th>
<th>ASCII Dec</th>
<th>ASCII Hex</th>
<th>EBCDIC Dec</th>
<th>EBCDIC Hex</th>
<th>NATIVE Dec</th>
<th>NATIVE Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>049</td>
<td>0</td>
<td>048</td>
<td>30</td>
<td>240</td>
<td>F0</td>
<td>048</td>
<td>30</td>
</tr>
<tr>
<td>050</td>
<td>1</td>
<td>049</td>
<td>31</td>
<td>241</td>
<td>F1</td>
<td>049</td>
<td>31</td>
</tr>
<tr>
<td>051</td>
<td>2</td>
<td>050</td>
<td>32</td>
<td>242</td>
<td>F2</td>
<td>050</td>
<td>32</td>
</tr>
<tr>
<td>052</td>
<td>3</td>
<td>051</td>
<td>33</td>
<td>243</td>
<td>F3</td>
<td>051</td>
<td>33</td>
</tr>
<tr>
<td>053</td>
<td>4</td>
<td>052</td>
<td>34</td>
<td>244</td>
<td>F4</td>
<td>052</td>
<td>34</td>
</tr>
<tr>
<td>054</td>
<td>5</td>
<td>053</td>
<td>35</td>
<td>245</td>
<td>F5</td>
<td>053</td>
<td>35</td>
</tr>
<tr>
<td>055</td>
<td>6</td>
<td>054</td>
<td>36</td>
<td>246</td>
<td>F6</td>
<td>054</td>
<td>36</td>
</tr>
<tr>
<td>056</td>
<td>7</td>
<td>055</td>
<td>37</td>
<td>247</td>
<td>F7</td>
<td>055</td>
<td>37</td>
</tr>
<tr>
<td>057</td>
<td>8</td>
<td>056</td>
<td>38</td>
<td>248</td>
<td>F8</td>
<td>056</td>
<td>38</td>
</tr>
<tr>
<td>058</td>
<td>9</td>
<td>057</td>
<td>39</td>
<td>249</td>
<td>F9</td>
<td>057</td>
<td>39</td>
</tr>
<tr>
<td>059</td>
<td>:</td>
<td>058</td>
<td>3A</td>
<td>122</td>
<td>7A</td>
<td>058</td>
<td>3A</td>
</tr>
<tr>
<td>060</td>
<td>;</td>
<td>059</td>
<td>3B</td>
<td>094</td>
<td>5E</td>
<td>059</td>
<td>3B</td>
</tr>
<tr>
<td>061</td>
<td>&lt;</td>
<td>060</td>
<td>3C</td>
<td>076</td>
<td>4C</td>
<td>060</td>
<td>3C</td>
</tr>
<tr>
<td>062</td>
<td>=</td>
<td>061</td>
<td>3D</td>
<td>126</td>
<td>7E</td>
<td>061</td>
<td>3D</td>
</tr>
<tr>
<td>063</td>
<td>&gt;</td>
<td>062</td>
<td>3E</td>
<td>110</td>
<td>6E</td>
<td>062</td>
<td>3E</td>
</tr>
<tr>
<td>064</td>
<td>?</td>
<td>063</td>
<td>3F</td>
<td>111</td>
<td>6F</td>
<td>063</td>
<td>3F</td>
</tr>
<tr>
<td>065</td>
<td>@</td>
<td>064</td>
<td>40</td>
<td>124</td>
<td>7C</td>
<td>064</td>
<td>40</td>
</tr>
<tr>
<td>066</td>
<td>A</td>
<td>065</td>
<td>41</td>
<td>193</td>
<td>C1</td>
<td>065</td>
<td>41</td>
</tr>
<tr>
<td>067</td>
<td>B</td>
<td>066</td>
<td>42</td>
<td>194</td>
<td>C2</td>
<td>066</td>
<td>42</td>
</tr>
<tr>
<td>068</td>
<td>C</td>
<td>067</td>
<td>43</td>
<td>195</td>
<td>C3</td>
<td>067</td>
<td>43</td>
</tr>
<tr>
<td>069</td>
<td>D</td>
<td>068</td>
<td>44</td>
<td>196</td>
<td>C4</td>
<td>068</td>
<td>44</td>
</tr>
<tr>
<td>070</td>
<td>E</td>
<td>069</td>
<td>45</td>
<td>197</td>
<td>C5</td>
<td>069</td>
<td>45</td>
</tr>
</tbody>
</table>

478
<table>
<thead>
<tr>
<th>Position</th>
<th>Character</th>
<th>ASCII Dec</th>
<th>ASCII Hex</th>
<th>EBCDIC Dec</th>
<th>EBCDIC Hex</th>
<th>NATIVE Dec</th>
<th>NATIVE Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>071</td>
<td>F</td>
<td>070</td>
<td>46</td>
<td>198</td>
<td>C6</td>
<td>070</td>
<td>46</td>
</tr>
<tr>
<td>072</td>
<td>G</td>
<td>071</td>
<td>47</td>
<td>199</td>
<td>C7</td>
<td>071</td>
<td>47</td>
</tr>
<tr>
<td>073</td>
<td>H</td>
<td>072</td>
<td>48</td>
<td>200</td>
<td>C8</td>
<td>072</td>
<td>48</td>
</tr>
<tr>
<td>074</td>
<td>I</td>
<td>073</td>
<td>49</td>
<td>201</td>
<td>C9</td>
<td>073</td>
<td>49</td>
</tr>
<tr>
<td>075</td>
<td>J</td>
<td>074</td>
<td>4A</td>
<td>209</td>
<td>D1</td>
<td>074</td>
<td>4A</td>
</tr>
<tr>
<td>076</td>
<td>K</td>
<td>075</td>
<td>4B</td>
<td>210</td>
<td>D2</td>
<td>075</td>
<td>4B</td>
</tr>
<tr>
<td>077</td>
<td>L</td>
<td>076</td>
<td>4C</td>
<td>211</td>
<td>D3</td>
<td>076</td>
<td>4C</td>
</tr>
<tr>
<td>078</td>
<td>M</td>
<td>077</td>
<td>4D</td>
<td>212</td>
<td>D4</td>
<td>077</td>
<td>4D</td>
</tr>
<tr>
<td>079</td>
<td>N</td>
<td>078</td>
<td>4E</td>
<td>213</td>
<td>D5</td>
<td>078</td>
<td>4E</td>
</tr>
<tr>
<td>080</td>
<td>O</td>
<td>079</td>
<td>4F</td>
<td>214</td>
<td>D6</td>
<td>079</td>
<td>4F</td>
</tr>
<tr>
<td>081</td>
<td>P</td>
<td>080</td>
<td>50</td>
<td>215</td>
<td>D7</td>
<td>080</td>
<td>50</td>
</tr>
<tr>
<td>082</td>
<td>Q</td>
<td>081</td>
<td>51</td>
<td>216</td>
<td>D8</td>
<td>081</td>
<td>51</td>
</tr>
<tr>
<td>083</td>
<td>R</td>
<td>082</td>
<td>52</td>
<td>217</td>
<td>D9</td>
<td>082</td>
<td>52</td>
</tr>
<tr>
<td>084</td>
<td>S</td>
<td>083</td>
<td>53</td>
<td>226</td>
<td>E2</td>
<td>083</td>
<td>53</td>
</tr>
<tr>
<td>085</td>
<td>T</td>
<td>084</td>
<td>54</td>
<td>227</td>
<td>E3</td>
<td>084</td>
<td>54</td>
</tr>
<tr>
<td>086</td>
<td>U</td>
<td>085</td>
<td>55</td>
<td>228</td>
<td>E4</td>
<td>085</td>
<td>55</td>
</tr>
<tr>
<td>087</td>
<td>V</td>
<td>086</td>
<td>56</td>
<td>229</td>
<td>E5</td>
<td>086</td>
<td>56</td>
</tr>
<tr>
<td>088</td>
<td>W</td>
<td>087</td>
<td>57</td>
<td>230</td>
<td>E6</td>
<td>087</td>
<td>57</td>
</tr>
<tr>
<td>089</td>
<td>X</td>
<td>088</td>
<td>58</td>
<td>231</td>
<td>E7</td>
<td>088</td>
<td>58</td>
</tr>
<tr>
<td>090</td>
<td>Y</td>
<td>089</td>
<td>59</td>
<td>232</td>
<td>E8</td>
<td>089</td>
<td>59</td>
</tr>
<tr>
<td>091</td>
<td>Z</td>
<td>090</td>
<td>5A</td>
<td>233</td>
<td>E9</td>
<td>090</td>
<td>5A</td>
</tr>
<tr>
<td>092</td>
<td>[</td>
<td>091</td>
<td>5B</td>
<td></td>
<td>091</td>
<td>5B</td>
<td></td>
</tr>
<tr>
<td>093</td>
<td>\</td>
<td>092</td>
<td>5C</td>
<td>224</td>
<td>E0</td>
<td>092</td>
<td>5C</td>
</tr>
<tr>
<td>094</td>
<td>]</td>
<td>093</td>
<td>5D</td>
<td></td>
<td>093</td>
<td>5D</td>
<td></td>
</tr>
<tr>
<td>095</td>
<td>^</td>
<td>094</td>
<td>5E</td>
<td>095</td>
<td>5F</td>
<td>094</td>
<td>5E</td>
</tr>
<tr>
<td>096</td>
<td>_</td>
<td>095</td>
<td>5F</td>
<td>109</td>
<td>6D</td>
<td>095</td>
<td>5F</td>
</tr>
<tr>
<td>097</td>
<td></td>
<td>096</td>
<td>60</td>
<td>121</td>
<td>79</td>
<td>096</td>
<td>60</td>
</tr>
<tr>
<td>098</td>
<td>a</td>
<td>097</td>
<td>61</td>
<td>129</td>
<td>81</td>
<td>097</td>
<td>61</td>
</tr>
<tr>
<td>099</td>
<td>b</td>
<td>098</td>
<td>62</td>
<td>130</td>
<td>82</td>
<td>098</td>
<td>62</td>
</tr>
<tr>
<td>100</td>
<td>c</td>
<td>099</td>
<td>63</td>
<td>131</td>
<td>83</td>
<td>099</td>
<td>63</td>
</tr>
<tr>
<td>101</td>
<td>d</td>
<td>100</td>
<td>64</td>
<td>132</td>
<td>84</td>
<td>100</td>
<td>64</td>
</tr>
<tr>
<td>102</td>
<td>e</td>
<td>101</td>
<td>65</td>
<td>133</td>
<td>85</td>
<td>101</td>
<td>65</td>
</tr>
<tr>
<td>103</td>
<td>f</td>
<td>102</td>
<td>66</td>
<td>134</td>
<td>86</td>
<td>102</td>
<td>66</td>
</tr>
<tr>
<td>104</td>
<td>g</td>
<td>103</td>
<td>67</td>
<td>135</td>
<td>87</td>
<td>103</td>
<td>67</td>
</tr>
<tr>
<td>105</td>
<td>h</td>
<td>104</td>
<td>68</td>
<td>136</td>
<td>88</td>
<td>104</td>
<td>68</td>
</tr>
</tbody>
</table>
### Appendix B. Character Sets

<table>
<thead>
<tr>
<th>Position</th>
<th>Character</th>
<th>ASCII</th>
<th>EBCDIC</th>
<th>NATIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>106</td>
<td>i</td>
<td>105</td>
<td>69</td>
<td>137</td>
</tr>
<tr>
<td>107</td>
<td>j</td>
<td>106</td>
<td>6A</td>
<td>145</td>
</tr>
<tr>
<td>108</td>
<td>k</td>
<td>107</td>
<td>6B</td>
<td>146</td>
</tr>
<tr>
<td>109</td>
<td>l</td>
<td>108</td>
<td>6C</td>
<td>147</td>
</tr>
<tr>
<td>110</td>
<td>m</td>
<td>109</td>
<td>6D</td>
<td>148</td>
</tr>
<tr>
<td>111</td>
<td>n</td>
<td>110</td>
<td>6E</td>
<td>149</td>
</tr>
<tr>
<td>112</td>
<td>o</td>
<td>111</td>
<td>6F</td>
<td>150</td>
</tr>
<tr>
<td>113</td>
<td>p</td>
<td>112</td>
<td>70</td>
<td>151</td>
</tr>
<tr>
<td>114</td>
<td>q</td>
<td>113</td>
<td>71</td>
<td>152</td>
</tr>
<tr>
<td>115</td>
<td>r</td>
<td>114</td>
<td>72</td>
<td>153</td>
</tr>
<tr>
<td>116</td>
<td>s</td>
<td>115</td>
<td>73</td>
<td>162</td>
</tr>
<tr>
<td>117</td>
<td>t</td>
<td>116</td>
<td>74</td>
<td>163</td>
</tr>
<tr>
<td>118</td>
<td>u</td>
<td>117</td>
<td>75</td>
<td>164</td>
</tr>
<tr>
<td>119</td>
<td>v</td>
<td>118</td>
<td>76</td>
<td>165</td>
</tr>
<tr>
<td>120</td>
<td>w</td>
<td>119</td>
<td>77</td>
<td>166</td>
</tr>
<tr>
<td>121</td>
<td>x</td>
<td>120</td>
<td>78</td>
<td>167</td>
</tr>
<tr>
<td>122</td>
<td>y</td>
<td>121</td>
<td>79</td>
<td>168</td>
</tr>
<tr>
<td>123</td>
<td>z</td>
<td>122</td>
<td>7A</td>
<td>169</td>
</tr>
<tr>
<td>124</td>
<td>{</td>
<td>123</td>
<td>7B</td>
<td>192</td>
</tr>
<tr>
<td>125</td>
<td></td>
<td></td>
<td>124</td>
<td>7C</td>
</tr>
<tr>
<td>126</td>
<td>}</td>
<td>125</td>
<td>7D</td>
<td>208</td>
</tr>
<tr>
<td>127</td>
<td>~</td>
<td>126</td>
<td>7E</td>
<td>161</td>
</tr>
<tr>
<td>128</td>
<td>DEL</td>
<td>127</td>
<td>7F</td>
<td>007</td>
</tr>
<tr>
<td>129</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>131</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>132</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>133</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>134</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>135</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>136</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>137</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>138</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>139</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>141</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Appendix B. Character Sets

<table>
<thead>
<tr>
<th>Position</th>
<th>Character</th>
<th>ASCII Dec</th>
<th>ASCII Hex</th>
<th>EBCDIC Dec</th>
<th>EBCDIC Hex</th>
<th>NATIVE Dec</th>
<th>NATIVE Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td></td>
<td>141</td>
<td>8D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td></td>
<td>142</td>
<td>8E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td></td>
<td>143</td>
<td>8F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>144</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td></td>
<td>145</td>
<td>91</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>146</td>
<td></td>
<td>146</td>
<td>92</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>147</td>
<td></td>
<td>147</td>
<td>93</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>148</td>
<td></td>
<td>148</td>
<td>94</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>149</td>
<td></td>
<td>149</td>
<td>95</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td>150</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>151</td>
<td></td>
<td>151</td>
<td>97</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>152</td>
<td></td>
<td>152</td>
<td>98</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>153</td>
<td></td>
<td>153</td>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>154</td>
<td></td>
<td>154</td>
<td>9A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>155</td>
<td></td>
<td>155</td>
<td>9B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>156</td>
<td></td>
<td>156</td>
<td>9C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>157</td>
<td></td>
<td>157</td>
<td>9D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>158</td>
<td></td>
<td>158</td>
<td>9E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>159</td>
<td></td>
<td>159</td>
<td>9F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td>160</td>
<td>A0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>161</td>
<td></td>
<td>161</td>
<td>A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>162</td>
<td></td>
<td>162</td>
<td>A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>163</td>
<td></td>
<td>163</td>
<td>A3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>164</td>
<td></td>
<td>164</td>
<td>A4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>165</td>
<td></td>
<td>165</td>
<td>A5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>166</td>
<td></td>
<td>166</td>
<td>A6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>167</td>
<td></td>
<td>167</td>
<td>A7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>168</td>
<td></td>
<td>168</td>
<td>A8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>169</td>
<td></td>
<td>169</td>
<td>A9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td></td>
<td>170</td>
<td>AA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>171</td>
<td></td>
<td>171</td>
<td>AB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>172</td>
<td></td>
<td>172</td>
<td>AC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>173</td>
<td></td>
<td>173</td>
<td>AD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>174</td>
<td></td>
<td>174</td>
<td>AE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>175</td>
<td></td>
<td>175</td>
<td>AF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>176</td>
<td></td>
<td>176</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position</td>
<td>Character</td>
<td>ASCII Dec</td>
<td>ASCII Hex</td>
<td>EBCDIC Dec</td>
<td>EBCDIC Hex</td>
<td>NATIVE Dec</td>
<td>NATIVE Hex</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>177</td>
<td></td>
<td></td>
<td></td>
<td>176</td>
<td>B0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>178</td>
<td></td>
<td></td>
<td></td>
<td>177</td>
<td>B1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>179</td>
<td></td>
<td></td>
<td></td>
<td>178</td>
<td>B2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
<td></td>
<td>179</td>
<td>B3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>181</td>
<td></td>
<td></td>
<td></td>
<td>180</td>
<td>B4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>182</td>
<td></td>
<td></td>
<td></td>
<td>181</td>
<td>B5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>183</td>
<td></td>
<td></td>
<td></td>
<td>182</td>
<td>B6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>184</td>
<td></td>
<td></td>
<td></td>
<td>183</td>
<td>B7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>185</td>
<td></td>
<td></td>
<td></td>
<td>184</td>
<td>B8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>186</td>
<td></td>
<td></td>
<td></td>
<td>185</td>
<td>B9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>187</td>
<td></td>
<td></td>
<td></td>
<td>186</td>
<td>BA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>188</td>
<td></td>
<td></td>
<td></td>
<td>187</td>
<td>BB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>189</td>
<td></td>
<td></td>
<td></td>
<td>188</td>
<td>BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td></td>
<td></td>
<td></td>
<td>189</td>
<td>BD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>191</td>
<td></td>
<td></td>
<td></td>
<td>190</td>
<td>BE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>192</td>
<td></td>
<td></td>
<td></td>
<td>191</td>
<td>BF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>193</td>
<td></td>
<td></td>
<td></td>
<td>192</td>
<td>C0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>194</td>
<td></td>
<td></td>
<td></td>
<td>193</td>
<td>C1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>195</td>
<td></td>
<td></td>
<td></td>
<td>194</td>
<td>C2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>196</td>
<td></td>
<td></td>
<td></td>
<td>195</td>
<td>C3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>197</td>
<td></td>
<td></td>
<td></td>
<td>196</td>
<td>C4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>198</td>
<td></td>
<td></td>
<td></td>
<td>197</td>
<td>C5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>199</td>
<td></td>
<td></td>
<td></td>
<td>198</td>
<td>C6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td>199</td>
<td>C7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>201</td>
<td></td>
<td></td>
<td></td>
<td>200</td>
<td>C8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>202</td>
<td></td>
<td></td>
<td></td>
<td>201</td>
<td>C9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>203</td>
<td></td>
<td></td>
<td></td>
<td>202</td>
<td>CA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>204</td>
<td></td>
<td></td>
<td></td>
<td>203</td>
<td>CB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>205</td>
<td></td>
<td></td>
<td></td>
<td>204</td>
<td>CC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>206</td>
<td></td>
<td></td>
<td></td>
<td>205</td>
<td>CD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>207</td>
<td></td>
<td></td>
<td></td>
<td>206</td>
<td>CE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>208</td>
<td></td>
<td></td>
<td></td>
<td>207</td>
<td>CF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>209</td>
<td></td>
<td></td>
<td></td>
<td>208</td>
<td>D0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td></td>
<td></td>
<td></td>
<td>209</td>
<td>D1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>211</td>
<td></td>
<td></td>
<td></td>
<td>210</td>
<td>D2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>212</td>
<td></td>
<td></td>
<td></td>
<td>211</td>
<td>D3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Appendix B. Character Sets

<table>
<thead>
<tr>
<th>Position</th>
<th>Character</th>
<th>ASCII</th>
<th>Dec</th>
<th>Hex</th>
<th>EBCDIC</th>
<th>Dec</th>
<th>Hex</th>
<th>NATIVE</th>
<th>Dec</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>213</td>
<td></td>
<td></td>
<td>212</td>
<td>D4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>214</td>
<td></td>
<td></td>
<td>213</td>
<td>D5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>215</td>
<td></td>
<td></td>
<td>214</td>
<td>D6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>216</td>
<td></td>
<td></td>
<td>215</td>
<td>D7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>217</td>
<td></td>
<td></td>
<td>216</td>
<td>D8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>218</td>
<td></td>
<td></td>
<td>217</td>
<td>D9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>219</td>
<td></td>
<td></td>
<td>218</td>
<td>DA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td></td>
<td></td>
<td>219</td>
<td>DB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>221</td>
<td></td>
<td></td>
<td>220</td>
<td>DC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>222</td>
<td></td>
<td></td>
<td>221</td>
<td>DD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>223</td>
<td></td>
<td></td>
<td>222</td>
<td>DE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>224</td>
<td></td>
<td></td>
<td>223</td>
<td>DF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>225</td>
<td></td>
<td></td>
<td>224</td>
<td>E0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>226</td>
<td></td>
<td></td>
<td>225</td>
<td>E1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>227</td>
<td></td>
<td></td>
<td>226</td>
<td>E2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>228</td>
<td></td>
<td></td>
<td>227</td>
<td>E3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>229</td>
<td></td>
<td></td>
<td>228</td>
<td>E4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td></td>
<td></td>
<td>229</td>
<td>E5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>231</td>
<td></td>
<td></td>
<td>230</td>
<td>E6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>232</td>
<td></td>
<td></td>
<td>231</td>
<td>E7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>233</td>
<td></td>
<td></td>
<td>232</td>
<td>E8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>234</td>
<td></td>
<td></td>
<td>233</td>
<td>E9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>235</td>
<td></td>
<td></td>
<td>234</td>
<td>EA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>236</td>
<td></td>
<td></td>
<td>235</td>
<td>EB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>237</td>
<td></td>
<td></td>
<td>236</td>
<td>EC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>238</td>
<td></td>
<td></td>
<td>237</td>
<td>ED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>239</td>
<td></td>
<td></td>
<td>238</td>
<td>EE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td></td>
<td></td>
<td>239</td>
<td>EF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>241</td>
<td></td>
<td></td>
<td>240</td>
<td>F0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>242</td>
<td></td>
<td></td>
<td>241</td>
<td>F1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>243</td>
<td></td>
<td></td>
<td>242</td>
<td>F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>244</td>
<td></td>
<td></td>
<td>243</td>
<td>F3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>245</td>
<td></td>
<td></td>
<td>244</td>
<td>F4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>246</td>
<td></td>
<td></td>
<td>245</td>
<td>F5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>247</td>
<td></td>
<td></td>
<td>246</td>
<td>F6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>248</td>
<td></td>
<td></td>
<td>247</td>
<td>F7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position</td>
<td>Character</td>
<td>ASCII</td>
<td>EBCDIC</td>
<td>NATIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-------</td>
<td>--------</td>
<td>--------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dec</td>
<td>Hex</td>
<td>Dec</td>
<td>Hex</td>
<td>Dec</td>
<td>Hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>249</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>248</td>
<td>F8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>249</td>
<td>F9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>251</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>250</td>
<td>FA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>252</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>251</td>
<td>FB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>253</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>252</td>
<td>FC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>254</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>253</td>
<td>FD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>255</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>254</td>
<td>FE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>255</td>
<td>FF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C. File Status Values

This appendix summarizes the values that can appear in FILE STATUS data items. The entry for each statement describes specific causes for each condition.

You may receive different file status values depending upon whether you use the standard compiler option with the v3 or 85 setting. Table C.1, “I-O File Status Values for the Default -std 85 Flag or /STANDARD=85 Qualifier Option” lists all file status values in numeric order for the default 85 setting. Table C.2, “I-O File Status Values for the V3 and 85 Options” lists the corresponding file status values for the v3 and 85 settings.

For more information about the standard compiler option, on a Tru64 UNIX system, refer to the COBOL man page. On an OpenVMS system, invoke the online help for COBOL.

Table C.1. I-O File Status Values for the Default -std 85 Flag or /STANDARD=85 Qualifier Option

<table>
<thead>
<tr>
<th>File Status</th>
<th>Input/Output Statements</th>
<th>File Organization</th>
<th>Access Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>Successful</td>
</tr>
<tr>
<td>02</td>
<td>REWRITE WRITE</td>
<td>Ind</td>
<td>All</td>
<td>Created duplicate alternate key</td>
</tr>
<tr>
<td>02</td>
<td>READ</td>
<td>Ind</td>
<td>All</td>
<td>Detected alternate duplicate key</td>
</tr>
<tr>
<td>04</td>
<td>READ</td>
<td>All</td>
<td>All</td>
<td>Record not size of user's buffer</td>
</tr>
<tr>
<td>05</td>
<td>OPEN</td>
<td>All</td>
<td>All</td>
<td>Optional file not present</td>
</tr>
<tr>
<td>07</td>
<td>CLOSE OPEN</td>
<td>All</td>
<td>All</td>
<td>Invalid file organization or device</td>
</tr>
<tr>
<td>10</td>
<td>READ</td>
<td>All</td>
<td>Seq</td>
<td>No next logical record or option file not present (at end)</td>
</tr>
<tr>
<td>14</td>
<td>READ</td>
<td>Rel</td>
<td>All</td>
<td>Relative record number too large</td>
</tr>
<tr>
<td>21</td>
<td>REWRITE</td>
<td>Ind</td>
<td>Seq</td>
<td>Primary key changed after READ</td>
</tr>
<tr>
<td>21</td>
<td>WRITE</td>
<td>Ind</td>
<td>Seq</td>
<td>Attempted nonascending key value (invalid key)</td>
</tr>
<tr>
<td>22</td>
<td>REWRITE</td>
<td>Ind</td>
<td>All</td>
<td>Duplicate alternate key (invalid key)</td>
</tr>
<tr>
<td>22</td>
<td>WRITE</td>
<td>Ind, Rel</td>
<td>Ran</td>
<td>Duplicate key (invalid key)</td>
</tr>
<tr>
<td>23</td>
<td>DELETE READ REWRITE START</td>
<td>Ind, Rel</td>
<td>Ran</td>
<td>Record not in file; optional file not present (invalid key)</td>
</tr>
</tbody>
</table>
### Appendix C. File Status Values

<table>
<thead>
<tr>
<th>File Status</th>
<th>Input/Output Statements</th>
<th>File Organization</th>
<th>Access Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>WRITE</td>
<td>Ind, Rel</td>
<td>All</td>
<td>Boundary violation or relative record number too large (invalid key)</td>
</tr>
<tr>
<td>30</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All other permanent errors</td>
</tr>
<tr>
<td>34</td>
<td>WRITE</td>
<td>Seq</td>
<td>Seq</td>
<td>Boundary violation</td>
</tr>
<tr>
<td>35</td>
<td>OPEN</td>
<td>All</td>
<td>All</td>
<td>File not found</td>
</tr>
<tr>
<td>37</td>
<td>OPEN</td>
<td>All</td>
<td>All</td>
<td>Inappropriate device type</td>
</tr>
<tr>
<td>38</td>
<td>OPEN</td>
<td>All</td>
<td>All</td>
<td>File previously closed with lock</td>
</tr>
<tr>
<td>39</td>
<td>OPEN</td>
<td>All</td>
<td>All</td>
<td>Conflict of file attributes</td>
</tr>
<tr>
<td>41</td>
<td>OPEN</td>
<td>All</td>
<td>All</td>
<td>File already opened</td>
</tr>
<tr>
<td>42</td>
<td>CLOSE</td>
<td>All</td>
<td>All</td>
<td>File not opened</td>
</tr>
<tr>
<td>43</td>
<td>DELETE REWRITE</td>
<td>All</td>
<td>Seq</td>
<td>No previous READ or START</td>
</tr>
<tr>
<td>44</td>
<td>REWRITE WRITE</td>
<td>All</td>
<td>All</td>
<td>Invalid record size</td>
</tr>
<tr>
<td>46</td>
<td>READ</td>
<td>All</td>
<td>Seq</td>
<td>No valid next record (at end)</td>
</tr>
<tr>
<td>47</td>
<td>READ START</td>
<td>All</td>
<td>All</td>
<td>File not open, or incompatible open mode</td>
</tr>
<tr>
<td>48</td>
<td>WRITE</td>
<td>All</td>
<td>All</td>
<td>File not open, or incompatible open mode</td>
</tr>
<tr>
<td>49</td>
<td>DELETE REWRITE</td>
<td>All</td>
<td>All</td>
<td>File not open, or incompatible open mode</td>
</tr>
<tr>
<td>90</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>Record locked by another user (record available)</td>
</tr>
<tr>
<td>91</td>
<td>OPEN</td>
<td>All</td>
<td>All</td>
<td>Open is unsuccessful; file locked by another access stream</td>
</tr>
<tr>
<td>92</td>
<td>DELETE READ REWRITE START WRITE</td>
<td>All</td>
<td>All</td>
<td>Record locked by another user (record not available)</td>
</tr>
</tbody>
</table>
### Appendix C. File Status Values

<table>
<thead>
<tr>
<th>File Status</th>
<th>Input/Output Statements</th>
<th>File Organization</th>
<th>Access Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>93</td>
<td>UNLOCK</td>
<td>All</td>
<td>All</td>
<td>No current record</td>
</tr>
<tr>
<td>94</td>
<td>UNLOCK</td>
<td>All</td>
<td>All</td>
<td>File not open, or incompatible open mode</td>
</tr>
<tr>
<td>95</td>
<td>OPEN</td>
<td>All</td>
<td>All</td>
<td>No file space on device</td>
</tr>
</tbody>
</table>

### Table C.2. I-O File Status Values for the V3 and 85 Options

<table>
<thead>
<tr>
<th>I-O Error Condition</th>
<th>Status Value</th>
<th>V3</th>
<th>85</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ successful – record shorter than fixed file attribute.</td>
<td>00</td>
<td>04</td>
<td></td>
</tr>
<tr>
<td>CLOSE reel/unit attempted on nonreel/unit device.</td>
<td>00</td>
<td>07</td>
<td></td>
</tr>
<tr>
<td>READ fails – relative key digits exceed relative key.</td>
<td>00</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>WRITE fails – relative key digits exceed relative key.</td>
<td>00</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>OPEN I-O on file that is not mass storage.</td>
<td>00</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>WRITE fails – attempt to write a record of a different size than in the file description.</td>
<td>00</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>READ fails – no next logical record (EOF detected).</td>
<td>13</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>READ fails – no next logical record (EOF on OPTIONAL file).</td>
<td>15</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>READ fails – no valid next record (already at EOF).</td>
<td>16</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>READ NEXT or sequential READ – no valid next record pointer.</td>
<td>16 1</td>
<td>46 1</td>
<td></td>
</tr>
<tr>
<td>READ or START fails – optional input file not present.</td>
<td>25</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>READ successful – record longer than fixed file attribute.</td>
<td>30</td>
<td>04</td>
<td></td>
</tr>
<tr>
<td>OPEN on relative or indexed file that is not mass storage.</td>
<td>30</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>REWRITE fails – attempt to rewrite record of different size.</td>
<td>30</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>CLOSE fails – file not currently open.</td>
<td>94</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>DELETE or REWRITE fails – previous I-O not successful READ.</td>
<td>93</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>OPEN fails – file previously closed with LOCK.</td>
<td>94</td>
<td>38</td>
<td></td>
</tr>
</tbody>
</table>
### Appendix C. File Status Values

<table>
<thead>
<tr>
<th>I-O Error Condition</th>
<th>Status Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN fails – file created with different organization.</td>
<td>V3 85</td>
</tr>
<tr>
<td>OPEN fails – file created with different prime record key.</td>
<td>94 39</td>
</tr>
<tr>
<td>OPEN fails – file created with different alternate record keys.</td>
<td>94 39</td>
</tr>
<tr>
<td>OPEN fails – file currently open.</td>
<td>94 41</td>
</tr>
<tr>
<td>READ or START fails – file not opened INPUT or I-O.</td>
<td>94 47</td>
</tr>
<tr>
<td>WRITE fails – file not opened OUTPUT, EXTEND, or I-O.</td>
<td>94 48</td>
</tr>
<tr>
<td>DELETE or REWRITE fails – file not opened I-O.</td>
<td>94 49</td>
</tr>
<tr>
<td>OPEN INPUT on a nonoptional file – file not found.</td>
<td>97 35</td>
</tr>
</tbody>
</table>

1Refer to the description of the /STANDARD qualifier in the COBOL online help file, or the VSI COBOL User Manual for the description of the `-std` flag, for information about the No Valid Next Record Condition.
Appendix D. Report Writer Presentation Rules and Tables

The tables and rules in this appendix specify the following:

- The permissible combinations of LINE NUMBER and NEXT GROUP clauses for each type of report group
- The requirements for the use of these clauses
- The interpretation that the Report Writer Control System (RWCS) gives to these clauses

D.1. Organization

There is an individual presentation rules table for each of the following types of report groups: REPORT HEADING, PAGE HEADING, PAGE FOOTING, and REPORT FOOTING. In addition, DETAIL report groups, CONTROL HEADING report groups, and CONTROL FOOTING report groups are treated jointly in the Body Group Presentation Rules Table.

Columns 1 and 2 of a presentation rules table list all of the permissible combinations of LINE NUMBER and NEXT GROUP clauses for the designated report group type. Consequently, to identify the set of presentation rules that applies to a particular combination of LINE NUMBER and NEXT GROUP clauses, read a presentation rules table from left to right along the selected row.

The applicable rules columns of a presentation rules table are divided into two parts. The first part specifies the rules that apply if the report description contains a PAGE clause, and the second part specifies the rules that apply if the PAGE clause is omitted. The explanation of the rules named in the applicable rules columns follows:

- Upper-limit rules and lower-limit rules:
  These rules specify the vertical subdivisions of the page within which the RWCS may present the specified report group when the PAGE clause is used.

- Fit test rules:
  The fit test rules are applicable only to body groups when the PAGE clause is included in the Report Description entry. Therefore, fit test rules are specified only within the Body Group Presentation Rules Table. The RWCS applies the fit test rules to determine whether the designated body group can be presented on the page on which the report is currently positioned.

- First print line position rules:
  The first print line position rules specify where on the page the RWCS presents the first print line of the given report group.

The presentation rules tables do not specify where on the page the RWCS presents the second and subsequent print lines (if any) of a report group; this is determined by the general rules of the LINE NUMBER clause.

- Next group rules:
  The next group rules relate to the proper use of the NEXT GROUP clause.

- Final LINE-COUNTER setting rules:
  These rules specify the values that the RWCS places in LINE-COUNTER after presenting report groups.

D.2. LINE NUMBER Clause Notation
Column 1 of the presentation rules table uses a shorthand notation to describe the sequence of LINE NUMBER clauses that may appear in the description of a report group. The meaning of the abbreviations used in column 1 is as follows:

1. The letter A represents one or more absolute LINE NUMBER clauses that appear in consecutive order within the sequence of LINE NUMBER clauses in the Report Group Description entry. None of the absolute LINE NUMBER clauses may have a NEXT PAGE phrase.

2. The letter R represents one or more relative LINE NUMBER clauses that appear in consecutive order within the sequence of LINE NUMBER clauses in the Report Group Description entry.

3. The letters NP represent one or more absolute LINE NUMBER clauses that appear in consecutive order within the sequence of LINE NUMBER clauses within the NEXT PAGE phrase appearing in the first, and only the first, LINE NUMBER clause.

4. When two abbreviations appear together, they refer to a sequence of LINE NUMBER clauses that consist of the two specified consecutive sequences. For example, A R refers to a Report Group Description entry within which the A sequence (defined in rule 1) is immediately followed by the R sequence (defined in rule 2).

5. A blank entry indicates that the clause is absent from the Report Group Description entry.

**D.3. LINE NUMBER Clause Sequence Substitutions**

Where A R is a permissible sequence in the presentation rules table, A is also permissible, and the same presentation rules apply.

Where NP R is a permissible sequence in the presentation rules table, NP is also permissible, and the same presentation rules apply.

**D.4. Saved-Next-Group-Integer Description**

Saved-next-group-integer is a data item that is addressable only by the RWCS. When an absolute NEXT GROUP clause specifies a vertical positioning value that cannot be accommodated on the current page, the RWCS stores that value in saved-next-group-integer. After page-advance processing, the RWCS positions the next body group using the value stored in saved-next-group-integer.

**D.5. REPORT HEADING Group Presentation Rules**

Figure D.1, “REPORT HEADING Group Presentation Rules” points to the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a REPORT HEADING report group.

**Figure D.1. REPORT HEADING Group Presentation Rules**

**REPORT HEADING Group Presentation Rules**

1. **Upper-limit rule:**

   The first line number on which the REPORT HEADING report group can be presented is the line number specified by the HEADING phrase of the PAGE clause.
2. Lower-limit rules:
   a. The last line number on which the REPORT HEADING report group can be presented is the line number that is obtained by subtracting 1 from the first-detail-line value of the FIRST DETAIL phrase of the PAGE clause.
   b. The last line number on which the REPORT HEADING report group can be presented is the line number specified by page-size of the PAGE clause.

3. First print line position rules:
   a. The first print line of the REPORT HEADING report group is presented on the line number specified by the integer of its LINE NUMBER clause.
   b. The first print line of the REPORT HEADING report group is presented on the line number obtained by adding the integer of the first LINE NUMBER clause and the value obtained by subtracting 1 from the heading-line value of the HEADING phrase of the PAGE clause.
   c. The REPORT HEADING report group is not presented.
   d. The first print line of the REPORT HEADING report group is presented on the line number obtained by adding the contents of its LINE-COUNTER (in this case, zero) to the integer of the first LINE NUMBER clause.

4. Next group rules:
   a. The NEXT GROUP integer must be greater than the line number on which the final print line of the REPORT HEADING report group is presented. In addition, the NEXT GROUP integer must be less than the line number specified by first-detail-line of the FIRST DETAIL phrase of the PAGE clause.
   b. The sum of the NEXT GROUP integer and the line number on which the final print line of the REPORT HEADING report group is presented must be less than the value of first-detail-line of the FIRST DETAIL phrase of the PAGE clause.
   c. NEXT GROUP NEXT PAGE signifies that the REPORT HEADING report group will appear by itself on the first page of the report. The RWCS processes no other report group while positioned at the first page of the report.

5. Final LINE-COUNTER setting rules:
   a. After the REPORT HEADING report group is presented, the RWCS places the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting.
   b. After the REPORT HEADING report group is presented, the RWCS places the sum of these two items into LINE-COUNTER as the final LINE-COUNTER setting:
      • The NEXT GROUP integer
      • The line number on which the final print line of the REPORT HEADING report group was presented
   c. After the REPORT HEADING report group is presented, the RWCS places zero into LINE-COUNTER as the final LINE-COUNTER setting.
   d. After the REPORT HEADING report group is presented, the final LINE-COUNTER setting is the line number on which the final print line of the REPORT HEADING report group was presented.
   e. LINE-COUNTER is unaffected by the processing of a nonprintable report group.

D.6. PAGE HEADING Group Presentation Rules
Appendix D. Report Writer
Presentation Rules and Tables

Figure D.2, “PAGE HEADING Group Presentation Rules Table” shows the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a PAGE HEADING report group.

**Figure D.2. PAGE HEADING Group Presentation Rules Table**

**PAGE HEADING Group Presentation Rules**

1. **Upper-limit rule:**
   
   If a REPORT HEADING report group has been presented on the page on which the PAGE HEADING report group is to be presented, then the first line number on which the PAGE HEADING report group can be presented is one greater than the final LINE-COUNTER setting established by the REPORT HEADING.

   Otherwise, the first line number on which the PAGE HEADING report group can be presented is the line number specified by the HEADING phrase of the PAGE clause.

2. **Lower-limit rule:**
   
   The last line number on which the PAGE HEADING report group can be presented is the line number obtained by subtracting 1 from the first-detail-line value of the FIRST DETAIL phrase of the PAGE clause.

3. **First print line position rules:**
   
   a. The first print line of the PAGE HEADING report group is presented on the line number specified by the integer of its LINE NUMBER clause.

   b. If a REPORT HEADING report group has been presented on the page on which the PAGE HEADING report group is to be presented, then the sum of the following two items defines the line number on which the first print line of the PAGE HEADING report group is presented:
      
      • The final LINE-COUNTER setting established by the REPORT HEADING report group
      • The integer of the first LINE NUMBER clause of the PAGE HEADING report group

   Otherwise, the sum of the following two items defines the line number on which the first print line of the PAGE HEADING report group is presented:
      
      • The integer of the first LINE NUMBER clause of the PAGE HEADING report group
      • The value obtained by subtracting 1 from the heading-line value of the HEADING phrase of the PAGE clause

   c. The PAGE HEADING report group is not presented.

4. **Final LINE-COUNTER setting rules:**
   
   a. The final LINE-COUNTER setting is the line number on which the final print line of the PAGE HEADING report group was presented.

   b. LINE-COUNTER is unaffected by the processing of a nonprintable report group.

**D.7. Body Group Presentation Rules**

Figure D.3, “Body Group Presentation Rules” points to the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in CONTROL HEADING, DETAIL, and CONTROL FOOTING report groups.

**Figure D.3. Body Group Presentation Rules**
Body Group Presentation Rules

1. Upper-limit rule:

   The first line number on which a body group can be presented is the first-detail-line value in the FIRST DETAIL phrase of the PAGE clause.

2. Lower-limit rules:

   The last line number on which a CONTROL HEADING report group or DETAIL report group can be presented is the last-detail-line value in the LAST DETAIL phrase of the PAGE clause.

   The last line number on which a CONTROL FOOTING report group can be presented is the line number specified by the footing-line value in the FOOTING phrase of the PAGE clause.

3. Fit test rules:

   a. If the value in LINE-COUNTER is less than the integer of the first absolute LINE NUMBER clause, then the body group will appear on the page on which the report is currently positioned.

      Otherwise, the RWCS executes page-advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS determines whether the saved-next-group-integer location was set when the final body group was presented on the preceding page. (See final LINE-COUNTER setting rule 6a.) If saved-next-group-integer was not set, the body group will be presented on the page on which the report is currently positioned. If saved-next-group-integer was so set, the RWCS:

         • Moves the saved-next-group-integer into LINE-COUNTER
         • Resets saved-next-group-integer to zero
         • Reapplies fit test rule 3a

   b. If a body group has been presented on the page on which the report is positioned, the RWCS computes a trial sum in a work location. The trial sum is computed by adding:

         • The contents of LINE-COUNTER
         • The integers of all LINE NUMBER clauses of the report group

      If the trial sum is not greater than the body group's lower-limit integer, then the report group is presented on the current page. If the trial sum exceeds the body group's lower-limit integer, then the RWCS executes page-advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS reapplies fit test rule 3b.

      If no body group has yet been presented on the page on which the report is currently positioned, the RWCS determines whether the saved-next-group-integer location was set when the final body group was presented on the preceding page. (See final LINE-COUNTER setting rule 6a.)

      If saved-next-group-integer was not set, the body group appears on the page on which the report is currently positioned.

      If saved-next-group-integer was set, the RWCS:

         • Moves the saved-next-group-integer into LINE-COUNTER
         • Resets saved-next-group-integer to zero
         • Computes a trial sum in a work location

      The trial sum is computed by adding:
• The contents of LINE-COUNTER

• The integer 1

• The integers of all but the first LINE NUMBER clause of the body group

If the trial sum is not greater than the body group's lower-limit integer, then the body group is presented on the current page. If the trial sum exceeds the body group's lower-limit integer, then the RWCS executes page-advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS presents the body group on that page.

c. If a body group has been presented on the page on which the report is currently positioned, the RWCS executes page-advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS reapplys Fit Test rule 3c.

If no body group has yet been presented on the page on which the report is currently positioned, the RWCS determines whether the saved-next-group-integer location was set when the final body group was presented on the preceding page. (See final LINE-COUNTER setting rule 6a). If saved-next-group-integer was not set, the body group will be presented on the page on which the report is currently positioned. If saved-next-group-integer was set, the RWCS moves saved-next-group-integer into LINE-COUNTER and resets saved-next-group-integer to zero.

If the value in LINE-COUNTER is less than the integer of the first absolute LINE NUMBER clause, the RWCS presents the body group on the page on which the report is currently positioned. Otherwise, the RWCS executes page-advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS presents the body group on that page.

4. First print line position rules:

a. The first print line of the body group appears on the line number specified by the integer of its LINE NUMBER clause.

b. The RWCS presents the first print line of the current body group on the line immediately following the line indicated by the value contained in LINE-COUNTER if these two conditions are true:

  • The value in LINE-COUNTER is equal to or greater than the line number specified by the first-detail-line value in the FIRST DETAIL phrase of the PAGE clause.

  • No body group has previously been presented on the page on which the report is currently positioned.

The RWCS presents the first print line of the current body group on the line that is obtained by adding the contents of LINE-COUNTER and the integer of the first LINE NUMBER clause of the current body group if these two conditions are true:

  • The value in LINE-COUNTER is equal to or greater than the line number specified by the first-detail-line value in the FIRST DETAIL phrase of the PAGE clause.

  • A body group has previously been presented on the page to which the report is currently positioned.

If the value in LINE-COUNTER is less than the line number specified by the first-detail-line value in the FIRST DETAIL phrase of the PAGE clause, then the RWCS presents the first print line of the body group on the line specified by the FIRST DETAIL phrase.

c. The body group is not presented.

d. The line number on which the RWCS presents the first print line is the sum of the contents of:

  • LINE-COUNTER

  • The integer of the first LINE NUMBER clause
5. Next group rule:

The integer of the absolute NEXT GROUP clause (next-group-line-num) must specify a line number that is:
(a) not less than that specified in the FIRST DETAIL phrase of the PAGE clause, and (b) not greater than that specified in the FOOTING phrase of the PAGE clause.

6. Final LINE-COUNTER setting rules:

a. If the body group that has just been presented is a CONTROL FOOTING report group and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

If the line number on which the final print line of the body group was presented is less than the integer of the NEXT GROUP clause, then the RWCS places the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting. If the line number on which the final print line of the body group was presented is equal to or greater than the integer of the NEXT GROUP clause, then the RWCS places the line number specified by the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER setting. In addition, the RWCS places the NEXT GROUP integer into the saved-next-group-integer location.

b. If the body group that has just been presented is a CONTROL FOOTING report group, and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

For all other cases the RWCS computes a trial sum in a work location. The trial sum is computed by adding:

• The integer of the NEXT GROUP clause

• The line number on which the final print line of the body group was presented

If the sum is less than the line number specified by the footing-line value in the FOOTING phrase of the PAGE clause, then the RWCS places that sum into LINE-COUNTER as the final LINE-COUNTER setting.

If the sum is equal to or greater than the line number specified by the footing-line value in the FOOTING phrase of the PAGE clause, then the RWCS places that line number into LINE-COUNTER as the final LINE-COUNTER setting.

c. The final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented if:

• The body group that has just been presented is a CONTROL FOOTING report group.

• The CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break.

For all other cases the RWCS places the line number specified by the footing-line value in the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER setting.

d. The final LINE-COUNTER setting is the line number on which the final print line of the body group was presented.

e. LINE-COUNTER is unaffected by the processing of a nonprintable body group.

f. The final LINE-COUNTER setting is the line number on which the RWCS presents the final print line of the CONTROL FOOTING report group if:

• The body group that has just been presented is a CONTROL FOOTING report group.

• The CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break.
For all other cases the RWCS uses the sum of these two items as the final LINE-COUNTER setting:

- The line number on which the final print line was presented
- The NEXT GROUP integer

### D.8. PAGE FOOTING Group Presentation Rules

Figure D.4, “PAGE FOOTING Group Presentation Rules” shows the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a PAGE FOOTING report group.

#### Figure D.4. PAGE FOOTING Group Presentation Rules

The PAGE FOOTING Group Presentation Rules are:

1. **Upper-limit rule:**
   
   The first line number on which the PAGE FOOTING report group can be presented is the line number obtained by adding:
   
   - The integer 1
   - The value of footing-line in the FOOTING phrase of the PAGE clause

2. **Lower-limit rule:**
   
   The last line number on which the PAGE FOOTING report group can be presented is the line number specified by page-size of the PAGE clause.

3. **First print line position rules:**
   
   a. The first print line of the PAGE FOOTING report group is presented on the line specified by the integer of its LINE NUMBER clause.
   
   b. The PAGE FOOTING report group is not presented.

4. **Next group rules:**
   
   a. The NEXT GROUP integer must be greater than the line number on which the final print line of the PAGE FOOTING report group is presented. In addition, the NEXT GROUP integer must not be greater than the line number specified by the page-size value of the PAGE clause.
   
   b. The sum of the following two items must not be greater than the line number specified by page-size of the PAGE clause:
      
      - The NEXT GROUP integer
      - The line number on which the final print line of the PAGE FOOTING report group is presented

5. **Final LINE-COUNTER setting rules:**
   
   a. The final LINE-COUNTER setting after the RWCS presents the PAGE FOOTING report group is the NEXT GROUP integer.
   
   b. The final LINE-COUNTER setting after the RWCS presents the PAGE FOOTING report group is the sum of:
      
      - The NEXT GROUP integer
• The line number on which the final print line of the PAGE FOOTING report group was presented.

c. After the PAGE FOOTING report group is presented, the final LINE-COUNTER setting is the line number on which the final print line of the PAGE FOOTING report group was presented.

d. LINE-COUNTER is unaffected by the processing of a nonprintable report group.

D.9. REPORT FOOTING Group Presentation Rules

Figure D.5, “REPORT FOOTING Group Presentation Rules” points to the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a REPORT FOOTING report group.

Figure D.5. REPORT FOOTING Group Presentation Rules

REPORT FOOTING Group Presentation Rules

1. Upper-limit rules:

   a. The first line number on which the REPORT FOOTING report group can be presented is one greater than the final LINE-COUNTER setting established by the PAGE FOOTING report group if a PAGE FOOTING report group has been presented on the page on which the report is positioned.

   Otherwise, the first line number on which the REPORT FOOTING report group can be presented is the line number obtained by adding 1 and the footing-line value of the PAGE clause.

   b. The first line number on which the REPORT FOOTING report group can be presented is the line number specified by the HEADING phrase of the PAGE clause.

2. Lower-limit rule:

   The last line number on which the REPORT FOOTING report group can be presented is the line number specified by the page-size value of the PAGE clause.

3. First print line position rules:

   a. The first print line of the REPORT FOOTING report group is presented on the line specified by the integer of its LINE NUMBER clause.

   b. If the RWCS presents a PAGE FOOTING report group on the page to which the report is positioned, then the sum of the following two items defines the line number on which the RWCS presents the first print line of the REPORT FOOTING report group:

      • The final LINE-COUNTER setting established by the PAGE FOOTING report group

      • The integer of the first LINE NUMBER clause of the REPORT FOOTING report group

   Otherwise, the sum of the following two items defines the line number on which the RWCS presents the first print line of the REPORT FOOTING report group:

      • The integer of the first LINE NUMBER clause of the REPORT FOOTING report group

      • The line number specified by the footing-line value of the FOOTING phrase of the PAGE clause

   c. The NEXT PAGE phrase in the first absolute LINE NUMBER clause directs the REPORT FOOTING report group to appear on a page on which no other report group has been presented. The first print line of the
REPORT FOOTING report group is presented on the line number specified by the integer of its LINE NUMBER clause.

d. The line number on which the RWCS presents the first print line is the sum of:
   
   • The contents of LINE-COUNTER
   
   • The integer of the first LINE NUMBER clause

e. The REPORT FOOTING report group is not presented.

4. Final LINE-COUNTER setting rules:

   a. The final LINE-COUNTER setting is the line number on which the RWCS presents the final print line of the REPORT FOOTING report group.

   b. LINE-COUNTER is unaffected by the processing of a nonprintable report group.
Appendix E. RTL Routines for Accessing the RAB and FAB Structures (OpenVMS Alpha and I64 Only)

In VSI COBOL for OpenVMS Alpha and OpenVMS I64, when a file is successfully opened, the file's RAB pointer is placed in its RMS_STV field and its FAB pointer is placed in the FABPTR field of the RAB. The two RTL routines documented here (DCOB$RMS_CURRENT_RAB and DCOB$RMS_CURRENT_FAB) enable VSI COBOL programmers to access the RAB and FAB data structures on OpenVMS Alpha and I64. However, the content and format of the RAB and FAB are not covered by any external standards (such as the ANSI standard for COBOL) and are subject to change.

Description

DCOB$RMS_CURRENT_FAB returns the address of the RMS FAB structure for the most recently used COBOL file connector. Some fields of the FAB are filled in by the RMS SYS$OPEN routine.

This routine can be used to obtain the address of the RMS FAB structure. The FAB is filled in by SYS$OPEN to reflect the actual attributes of the file in the cases where the actual attributes differ from the attributes specified by the COBOL file connector.

The FAB is a structure used internally by the VSI COBOL run-time system to implement COBOL semantics. Modification of the FAB can result in abnormal program behavior, including unexpected program termination.

Example

See the example for DCOB$RMS_CURRENT_RAB.

Description

DCOB$RMS_CURRENT_RAB returns the address of the RMS RAB structure for the most recently used COBOL file connector.

This routine can be used to obtain the address of the RMS RAB structure. The RAB describes attributes of the connection to a file.

The RAB is a structure used internally by the VSI COBOL run-time system to implement COBOL semantics. Modification of RAB fields can result in abnormal program behavior, including unexpected program termination.

Example

1. *+
   *  PROGRAM : RMSEXAMPLE
   *    PROGRAM DESCRIPTION:
   *    * This program is an example of use of DCOB$RMS_CURRENT_FAB
   *         and DCOB$RMS_CURRENT_RAB.
   * -
   IDENTIFICATION DIVISION.
   PROGRAM-ID. RMSEXAMPLE.
ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT EMPLOYEE-FILE
   ASSIGN TO "employee.dat"
   ORGANIZATION IS SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD EMPLOYEE-FILE
   BLOCK CONTAINS 2048 CHARACTERS.
01 EMPLOYEE.
   03 NAME PIC X(30).
   03 OFFICE PIC X(10).
   03 PHONE PIC X(10).

WORKING-STORAGE SECTION.
01 EMPLOYEE-FAB USAGE IS POINTER.
01 EMPLOYEE-RAB USAGE IS POINTER.

PROCEDURE DIVISION.
P0.
   * Open the file to establish EMPLOYEE-FILE as the current file.
   * OPEN INPUT EMPLOYEE-FILE.

   * Get the pointer to the RMS FAB structure for EMPLOYEE-FILE. Store
   * the pointer in EMPLOYEE-FAB. Do the same for the RAB.
   * CALL "DCOB$RMS_CURRENT_FAB" GIVING EMPLOYEE-FAB.
     CALL "DCOB$RMS_CURRENT_RAB" GIVING EMPLOYEE-RAB.

   * Pass the address of the FAB to a subroutine that will use
   * the contents.
   * CALL "PRINT-FIELDS" USING BY VALUE EMPLOYEE-FAB EMPLOYEE-RAB.

   * CLOSE the file before exiting.
   * CLOSE EMPLOYEE-FILE.
     STOP RUN.
END PROGRAM RMSEXAMPLE.

IDENTIFICATION DIVISION.
PROGRAM-ID. PRINT-FIELDS.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 TEMP-W.
   03 TEMP-WORD PIC S9(4) COMP.
01 TEMP-W1 REDEFINES TEMP-W.
   03 TEMP-B1 PIC X.
   03 TEMP-B2 PIC X.
Appendix E. RTL Routines
for Accessing the RAB and
FAB Structures (OpenVMS
Alpha and I64 Only)

LINKAGE SECTION.

01 FAB.
  05 FAB_ID   PIC S9(9) COMP VALUE 20483.
  05 FOP      PIC S9(9) COMP.
  05 STS      PIC S9(9) COMP.
  05 STV      PIC S9(9) COMP.
  05 ALQ      PIC S9(9) COMP.
  05 DEQ      PIC S9(4) COMP.
  05 FAC      PIC X.
  05 SHR      PIC X.
  05 CTX      PIC S9(9) COMP.
  05 RTV      PIC X.
  05 ORG      PIC X.
  05 RAT      PIC X.
  05 RFM      PIC X.
  05 JNL      PIC S9(9) COMP.
  05 XAB-ADD  USAGE IS POINTER.
  05 NAM-ADD  USAGE IS POINTER.
  05 FNA      USAGE IS POINTER.
  05 DNA      USAGE IS POINTER.
  05 FNS      PIC X.
  05 DNS      PIC X.
  05 MRS      PIC S9(4) COMP.
  05 MRN      PIC S9(9) COMP.
  05 BLS      PIC S9(4) COMP.
  05 BLN      PIC X.
  05 FILLER   PIC X(18).

01 RAB.
  05 RAB_ID   PIC S9(9) COMP VALUE 17409.
  05 ROP      PIC S9(9) COMP.
  05 STS      PIC S9(9) COMP.
  05 STV      PIC S9(9) COMP.
  05 RFA      PIC S9(4) COMP OCCURS 3 TIMES.
  05 RESERVED PIC S9(4) COMP.
  05 CTX      PIC S9(9) COMP.
  05 RAC      PIC X.
  05 TMO      PIC X.
  05 USZ      PIC S9(4) COMP.
  05 RSZ      PIC S9(4) COMP.
  05 UBF      PIC S9(9) COMP.
  05 RBF      PIC S9(9) COMP.
  05 RHB      PIC S9(9) COMP.
  05 KBF      PIC S9(9) COMP.
  05 KSZ      PIC X.
  05 KRF      PIC X.
  05 MBF      PIC X.
  05 MBC      PIC X.
  05 BKT      PIC S9(9) COMP.
  05 FABPTR   PIC S9(9) COMP.
  05 XAB      PIC S9(9) COMP.

PROCEDURE DIVISION USING FAB RAB.
  *
  * Convert the MBF PIC X value to a PIC S9(5) COMP VALUE.
  *
        MOVE 0 TO TEMP-WORD.
        MOVE MBF TO TEMP-B1.
  *


* Display the multibuffer count and file allocation. *

    DISPLAY "Multibuffer count is "
    TEMP-WORD WITH CONVERSION " blocks".
    DISPLAY "File allocation is " ALQ WITH CONVERSION.

    EXIT PROGRAM.
END PROGRAM PRINT-FIELDS.

Output:

2. $
$ LINK RMSEXAMPLE
$ RUN RMSEXAMPLE
File allocation is  1206
$
Index

Symbols
- nationality japan, 14
/ARITHMETIC=STANDARD qualifier , 222
/CHECK=BOUNDS qualifier, 208
/CHECK=DUPLICATE_KEYS qualifier, 115,
/INCLUDE qualifier , 445
/MATH_INTERMEDIATE=CIT3 qualifier , 224
/MATH_INTERMEDIATE=FLOAT qualifier , 223
/NATIONALITY=JAPAN, 14, 151
/RESERVED_WORDS=200X , 465
/RESERVED_WORDS=[NO]XOPEN , 465
/RESERVED_WORDS=FOREIGN_EXTENSIONS ,

A
Abbreviated combined relation conditions, 233-233
definition, 233
ACCEPT statement , 249-269
reference to devices, 47
ACOS intrinsic function, 406
ADD statement, 269-271
Additional reference
definition, 23
Alignment and padding, 93
Alignment, effect of SYNC clause, 175-176
ALL literal, 15
Alpha data alignment, 93
ALPHABET clause, 47
Alphabet-name
definition, 3
ALPHABETIC test, 227
alt-key, 114
ALTER statement, 271
ALTERNATE RECORD KEY clause
-relax_key_checking, 115
ANNUITY intrinsic function, 406
ANSI source reference format
definition, 18
APPLY clause, 79-80
ARGCOUNT intrinsic function, 407
Arithmetic
standard, 222
Arithmetic expressions, 220-224
composition of, 220
data items in, 221
definition, 220
evaluation of, 221
literals in, 221
operators in, 221
use of parentheses in, use of sign in, 221
Arithmetic operations
multiple receiving fields, 235
restrictions for operands, 235
rounding off results, 235
storing partial results, 235
Arithmetic operators, 221
definition, 221
Arithmetic statements
definition, 235
Arithmetic, native and standard, 221, 223
ASCENDING clause, 115
ASCII character set, , 477
ASIN intrinsic function, 407
At end condition, 242
definition, 242
ATAN intrinsic function, 408
AUTHOR paragraph, 31-31
Automatic record locking, 330

B
BELL clause, 117
Binary search
of a table,
BINARY usage, 181
BLANK clause, 117
Blank lines, 20
BLANK WHEN ZERO clause, 118
BLANK WHEN ZEROES, 118
Blanks
in file specification, 68
BLINK clause, 119
Block
definition, 86
related to a record, 86
Body Group Presentation Rules, table
, 492
Boundary equivalence, 88-93
Bucket
definition, 86
related to physical record, 86

C
CALL statement, 272-276
Called programs
Linkage Section of, 96
Procedure Division header of, 96
CANCEL statement, 276
Categories of data items, 87
CHAR intrinsic function, 409
Character sets
and collating sequence, 48
ASCII, 47, , 477
COBOL, 1
computer, 1
EBCDIC, 477
in ALPHABET clause, 47
Native, 477
Character transfer
using the STRING statement, 374-378
using the UNSTRING statement, 385-390
character-string
  definition, 2
Class condition, 227-228
  definition, 227
Class-name
  definition, 3
CLOSE statement, 278-280
cobcall routine, 276
cobcancel routine,
cobfunc routine,
COBOL character set
  definition, 1
COBOL language elements, 1
COBOL word
  definition, 2
CODE clause, 119
Collating sequence
  in ALPHABET clause, 47
  specifying in a COBOL program, 41
  when merging files, 317
Colon, 17
COLUMN NUMBER clause, 120
Combined condition
  definition, 231
Combining files (see Merging files)
Comma, as a separator, 16, 401
Command line arguments , 249
  syntax, 44
Comment lines, 20
COMP SYNC data items, 175-176
Comparing operands, 225
  when alphabetic, 227
  when nonnumeric, 226
  when numeric, 226, 227
Compatibility
  /CHECK=DUPPLICATE, 115, 162
  ASSIGN TO unquoted string, 67
case sensitivity, 68
descriptor, 275
parameter passing,
  spaces in file specs, 68
tabs in file specs, 68
VALUE OF ID environment variable,
VALUE OF ID logical, 195
Compiler options
  -nationality japan, 14
  /NATIONALITY=JAPAN, 14
Compiler-directing sentence, 202
Compiler-directing statement, 202
definition, 197
Complex condition, 231
definition, 224
COMPUTATIONAL usage,
COMPUTATIONAL-1 usage,
COMPUTATIONAL-2 usage,
COMPUTATIONAL-3 usage,
COMPUTATIONAL-5 usage,
COMPUTATIONAL-x usage,
COMPUTE statement, 280-282
Condition-name
  condition
  definition, 228
Condition-name condition, 228-228
Condition-names
  associating values with, 85, 189-195
definition, 3
  in general formats and rules, 211
  in SWITCH clause, 47
  qualifying, 211
Conditional compilation lines, 19, 20
Conditional expressions, 224-229
  class condition, 227-228
  combining, 231
  complex conditions, 231
  condition-name, 228-228
  definition, 224
  evaluation of, 233
  negating, 231
  relation condition, 225-227
  sign condition, 229
  success/failure condition, 229
  switch-status condition, 228-229
Conditional sentence, 202
Conditional statement, 202
definition, 197
Conditional variables
  definition, 228
  relation to condition-names, 3
Configuration Section
  definition, 39
CONTINUE statement, 282
CONTROL clause, 122
COPY statement, 442
CORRESPONDING phrase
  for arithmetic statements, 237
  for the MOVE statement, 237
COS intrinsic function, 409
Counting characters in a data item, 311-316
CRT status
  specifying CRT status, 51
  CRT STATUS IS clause, 51
  CURRENT DATE clause, 49
  CURRENT DATE intrinsic function, 410
  Cursor
    specifying cursor address, 50
    CURSOR IS clause, 50
D
Data description entry
  definition, 83
elements of, 83
Data Division , 94
  entries, 29
Data items
  assigning initial values to, 189-195
categories of, 87, 151
characteristics of, 149-158
classes of, 87
contents and class incompatibility, 238
default initial values, 96
default USAGE of, 220
in arithmetic expressions, 221
index, 208
naming, 125-126
nonstandard data positioning in, 134
storage format for, 180-189
DATA RECORDS clause, 126
Data transfer
positioning rules for, 87
using the MOVE statement, 319-324
Data-handling operations
undefined results
from incompatible data, 238
from operand overlap, 238
Data-name
definition, 3
in an identifier, 210-211
Data-name clause, 125-126
DATE-COMPILED paragraph, 31
DATE-OF-INTEGER intrinsic function, 411
DATE-TO-YYYYMMDD intrinsic function, 411
DAY-OF-INTEGER intrinsic function, 412
DAY-TO-YYYYYDDD intrinsic function, 413
Decimal point
specifying as comma, 50
DECIMAL-POINT IS COMMA clause, 50
Declarative procedures
referencing with the USE statement, 390-393
Declaratives
definition, 30
DEFERRED-WRITE, 78
DEFERRED-WRITE phrase of the APPLY clause, 79
DELETE statement, 282-284
Delimited-scope statement, 203
definition, 197
DESCENDING clause, 115
DISPLAY statement, 284-293
reference to devices, 47
usage,
DIVIDE statement, 294-296
Division by zero, 236
Division header
definition, 27
Divisions
definition, 1
Duplicate keys check, 115,
DUPLICATES phrase, 115

E
EBCDIC character set, 477
Editing rules
for PICTURE clause, 154
Editing symbols
in PICTURE clause, 151-154
Elementary data items
alternative groupings of, 165-167
definition, 83
Embedded spaces
in file specification, 68
END PROGRAM header, 399-400
ENVIRONMENT DIVISION
syntax and general rules, 39-39
Environment variables, 249
ERASE clause, 127
Error handling
with the USE statement, 390-393
EVALUATE statement, 297
Example
definition, 23
EXIT PROGRAM statement, 302
EXIT statement, 301
Exponentiation, 222
results when invalid, 236
EXTENSION phrase of the APPLY clause, 79
EXTERNAL clause, 127
External data
definition, 204

F
FAB (see File access block)
FACTORIAL intrinsic function, 414
FD entries, 95
Figurative constants, 14-15
definition, 14
HIGH-VALUE, 14
LOW-VALUE, 15
QUOTE, 15
SPACE, 14
symbolic-character, 15
File access, 326-333
File access block (FAB), 81-
File connector
definition, 3
File description
clauses of, 95
function of entry, 83
structure of, 95
File formats
Indexed, 393, 397
Line Sequential, 393, 395
Relative, 393, 396
Sequential, 393, 395
File mapping, 80
File Section of Data Division, 95
File specification
assigning to a file-name
using VALUE OF ID clause, 195
Spaces in, 68
Tabs in, 68
File status
/STANDARD=85 values, 485
/STANDARD=V3 values, 485
data item, 238
values, 485-488
FILE STATUS clause, 129
File-name
definition, 3
Files
Indexed, 278
Line Sequential, 278
Relative, 278
Sequential, 278
Fill bytes, def, 88
FILL-SIZE phrase of the APPLY clause, 79
FILLER, 125-126
Fixed insertion editing, 155
Fixed segment
definition, 245
Fixed-length records, 159-161
Floating insertion editing, 155
Floating-point literal
definition, 11
FOREGROUND-COLOR clause, 130
Format
of print files, 136-138
record (RECORD clause), 159-161
FROM phrase, 244
FULL clause, 131
Function-identifier, 401
Function-name, 401
Function-names, 10
Functions
definition, 23
Functions, intrinsic (see Intrinsic functions)

G
G_FLOATING, 184
General format
definition, 23
function of, 23
General rule
definition, 23
GENERATE statement, 303
Generic term
definition, 23
GLOBAL clause, 131
Global data
definition, 204
GO TO statement, 304-305
Group data item, 83
definition, 83
GROUP INDICATE clause, 132
Group moves, 322

H
Hexadecimal literal
definition, 13
HIGH-VALUE figurative constant, 14, 48
HIGHLIGHT clause, 133
Horizontal tab, 17

I
I-O File status values, 485
/STANDARD=85 qualifier (tab.), 485
/STANDARD=85 values, 485
/STANDARD=V3 qualifier (tab.), 487
/STANDARD=V3 values, 485
I-O status (see Input-output)
I64 data alignment, 93
IDENT clause, 33
Identification Division
syntax and general rules for, 33
Identifier, defined, 210-211
IF statement, 306-308
Imperative sentence, 202
Imperative statement, 202
definition, 197
Indentation, relation to level-numbers, 84
Independent segment
definition, 245
Index data item, 208
comparing, 227
defining in program, 183
INDEX SYNC data items, 175-176
INDEX usage,
Index-names
comparing, 227
definition, 3
rules, 144
storing value of in a data item, 208
Indexed files, 278, 393
duplicate key, 115,
starting position in, 369-373
Indexes
setting values for, 352, 357-361
Indexing, 208-209
basis for, 143
in an identifier, 211
Initialization of data item values
in Linkage Section, 96
in Working-Storage Section, 96
INITIALIZE statement, 308
with group and elementary items, 309
INITIATE statement, 310
Input-output
of low-volume data, 249, 284
status, 238-242
values for, 239
INPUT-OUTPUT Section
definition, 55
INSPECT statement, 311-316
INTEGER intrinsic function, 414
INTEGER-OF-DATE intrinsic function, 415
INTEGER-OF-DAY intrinsic function, 416
INTEGER-PART intrinsic function, 416
Intermediate data item
definition, 235
INTO phrase, of I-O statements, 244-245
Intrinsic functions, 401-440
  ACOS, 406
  ANNUITY, 406
  ARGCOUNT, 407
  ASIN, 407
  ATAN, 408
  CHAR, 409
  COS, 409
  CURRENT-DATE, 410
date manipulation, 401
  DATE-OF-INTEGER, 411
  DATE-TO-YYYYMMDD, 411
  DAY-OF-INTEGER, 412
  DAY-TO-YYYYDDD, 413
  FACTORIAL, 414
  function types
    alphanumeric, 401
    integer, 401
    numeric, 401
  INTEGER, 414
  INTEGER-OF-DATE, 415
  INTEGER-OF-DAY, 416
  INTEGER-PART, 416
  LENGTH, 417
  LOG, 418
  LOG10, 418
  LOWER-CASE, 419
  MAX, 419
  MEDIAN, 422
  MIDRANGE, 422
  MIN, 423
  MOD, 424
  NUMVAL, 424
  NUMVAL-C, 425
  ORD, 426
  ORD-MAX, 427
  ORD-MIN, 428
  other, 401
  PRESENT-VALUE, 428
  RANDOM, 429
  RANGE, 430
  relational, 401
  REM, 431
  REVERSE, 432
  scientific/mathematical, 401
  SIN, 432
  SQRT, 433
  STANDARD-DEVIVATION, 433
  statistical/accounting, 401
  string manipulation, 401
  SUM, 434
table, 403
  TAN, 435
  TEST-DATE-YYYYMMDD, 435
  TEST-DAY-YYYYDDD, 436
  UPPER-CASE, 437
  VARIANCE, 438
  WHEN-COMPILED, 438
  YEAR-TO-YYYY, 439
Invalid key condition, 243

J
  JUSTIFIED clause, 134
    and Standard Alignment Rules, 88

K
  Keyword
    definition, 6

L
  LABEL RECORDS clause, 135
  LENGTH intrinsic function, 417
Level-numbers, 84, 135-136
  01 to 49, 84
  definition, 3
  for records, 84
  LIB$ESTABLISH routine, 275
  LIB$REVERT routine,
    Library text, copying into source program, 442
  Library-name
    definition, 3
  LINAGE clause, 136-138
  LINAGE-COUNTER, 137
  Line continuation, 19
  LINE NUMBER clause, 139
    Report Writer, 489
    sequence substitutions, 490
  Line Sequential files, 278, 393
  I-O mode, 349
    REWRITE, 349
  Linear search of a table, 353
  Linkage Section, 96
  Literals, 10-13
    definition, 10
    in arithmetic expressions, 221
    numeric literals
      scientific notation, 11
  Location equivalence, 88-93
  LOCK-HOLDING phrase of the APPLY clause, 79
  Locked records, deleting, 283
  Locking operations on files, 278-280
  Locking records, 79
  LOG intrinsic function, 418
  LOG10 intrinsic function, 418
  Logical data
    characteristics, 83
  LOW-VALUE
    figurative constant, 15, 48
  LOWER-CASE intrinsic function, 419
  Lowercase letters, compiler treatment of, 2
  LOWLIGHT clause, 141
Magnetic tapes
   REEL, 66
   UNIT, 66
Major-minor equivalence technique
   definition, 88
Major-minor storage allocation, 88-93
Manual record locking,
MASS-INSERT phrase of the APPLY clause, 80
MATH_INTERMEDIATE=CIT4 qualifier, 224
MAX intrinsic function, 419
MEAN intrinsic function, 421
MEDIAN intrinsic function, 422
MEMORY SIZE clause of OBJECT-COMPUTER paragraph, 41
Memory size, documenting in a COBOL program,
MERGE statement, 316-319
Merging files, 316-319
   record transfer using the RETURN statement, 347-348
MIDRANGE intrinsic function, 422
Millennium date change
   ACCEPT statement, 252
   CURRENT-DATE function, 410
MIN intrinsic function, 423
MOD intrinsic function, 424
MOVE statement, 319-324
MULTIPLE FILE clause
   I-O-CONTROL paragraph, 80
   syntax rules for, 78
Multiple receiving fields
   definition, 235
Multiple record definitions, 135
Multiple results (see Multiple receiving fields)
MULTIPLY statement, 324-325
Name scoping
   definition, 204
Naming
   a COBOL program, 31-31
   files in a COBOL program, 61
National literals, 14
National literals (ex.), 14
Native character set, 477
Negated combined condition
   definition, 232
Negated simple condition
   definition, 231
Next executable statement
   definition, 220
NEXT GROUP clause, 142
Nonnumeric data transfer, 319-324
Nonnumeric literal
   definition, 12
NOT ON SIZE ERROR phrase
   description of, 235
Numeric data item
   maximum number of digit positions, 151
Numeric data transfer, 319-324
Numeric edited data item
   maximum number of digit positions, 151
Numeric literals
   definition, 10
   scientific notation, 11
NUMERIC test, 227
NUMVAL intrinsic function, 424
NUMVAL-C intrinsic function, 425
OBJECT-COMPUTER paragraph, 14
OCCURS clause, 143-146
   related to subscripting, 207
ON EXCEPTION phrase
   for ACCEPT statement, 237
   for CALL statement, 237
ON SIZE ERROR phrase
   description of, 235
OPEN statement, 326-333
   effect on LINAGE values, 137-
Optional word
   definition, 6
Oracle CDD/Repository Data Types and VSI COBOL Equivalents, 446
ORD intrinsic function, 426
ORD-MAX intrinsic function, 427
ORD-MIN intrinsic function, 428
Overlapping operands, 238
PACKED-DECIMAL usage, 181
Padding
   for Alpha alignment, 93
   for I64 alignment, 93
PAGE clause, 146
PAGE FOOTING Group Presentation Rules, table
   496
PAGE HEADING Group Presentation Rules, table, 491
PAGE-COUNTER option
   definition, 7
   Paragraph, 29
   definition, 29
   Paragraph header
   definition, 29
   Paragraph-name, 29
Parentheses, 16
   in arithmetic expressions, 221, 221
PERFORM statement, 333-340
Period
   as separator, 16
Physical data
   characteristics, 83
PICTURE character-strings, 15
PICTURE clause, 15, 149-158  
currency symbol, 49  
editing methods, 154-157  
symbol precedence rules, 157  
POINTER SYNC data items, 175-176  
POINTER usage,  
POINTER-64 , 189  
Preallocation of disk blocks, 80  
PREALLOCATION phrase of the APPLY clause,  
PRESENT-VALUE intrinsic function , 428  
PRINT command  
/NOFEED qualifier,  
for LINAGE files, 138  
Print file  
format, 80, 136-138  
PRINT-CONTROL phrase of the APPLY clause,  
Procedure Division  
transfer of control within, 219-220  
USING phrase, 246  
Program execution  
terminating with STOP statement, 373  
PROGRAM-ID paragraph, 31-31  
Pseudo-text , 20  
delimiter, 17  
Q  
Qualification, 204-206  
definition, 204  
in an identifier, 211  
Qualifiers  
/INCLUDE,  
definition, 204  
Quotation marks, 17  
QUOTE figurative constant, 15  
R  
RAB (see Record access block)  
RANDOM intrinsic function , 429  
RANGE intrinsic function , 430  
RD entries, 96  
READ statement, 340-345  
Record  
as a logical concept, 83  
as a physical concept, 83  
defining length of, 159-161  
deleting from files, 282-284  
when manually locked, 283  
unit of transfer for, 86  
Record access  
using the READ statement, 340-345  
using the RELEASE statement, 346-347  
using the RETURN statement, 347-348  
using the START statement, 369-373  
Record access block (RAB),  
Record allocation, 88-93  
RECORD clause, 159-161  
Record description  
function of entry, 83  
hierarchical structure, 84  
level-numbers, 84  
RECORD KEY clause, 161  
Record locking  
automatic,  
manual,  
RECORD statement, 345-346  
Record transfer  
using the WRITE statement, 393-399  
REDEFINES clause, 162-165  
Reference modification, 209  
Relation condition, 225-227  
definition, 225  
Relational operators  
definitions, 225  
Relative files, 278, 393  
starting position in, 369-373  
Release notes  
where to find, viii  
RELEASE statement, 346-347  
REM intrinsic function , 431  
Removal operations for file media, 278-280  
RENAMES clause, 165-167  
REPLACE statement, 456-460  
Replacing characters in a data item, 311-316  
Replacing records (with REWRITE statement), 348-351  
REPORT clause, 167  
Report description entry, 168  
REPORT FOOTING Group Presentation Rules, table , 497  
REPORT HEADING Group Presentation Rules, table, 490  
Report Section of Data Division, 96  
Report Writer  
organization, 489  
Report Writer presentation rules, tables, 489  
REQUIRED clause, 168  
Required words , 6-6  
definition, 6  
RERUN clause of the I-O-CONTROL paragraph  
general rules for, 80  
syntax rules for, 78  
Reserved words , 6-10  
definition, 6  
list of, 461  
Resultant identifiers  
definition, 235  
function of, 235  
RETURN key, to shorten source lines, 20  
RETURN statement, 347-348  
RETURN-CODE special register, 248, 275  
REVERSE intrinsic function , 432  
REVERSE-VIDEO clause, 169  
Rewind operations for file media, 278-280
REWRITE statement, 348-351
RMS
  file extension, 79
Rounding off arithmetic results, 235

S
SAME AREA clause
  I-O-CONTROL paragraph
    general rules for, 80
    syntax rules for, 78
SAME RECORD AREA clause of the I-O-CONTROL paragraph, 80
SAME SORT AREA clause of the I-O-CONTROL paragraph, 80
Saved-Next-Group-Integer
  Report Writer, 490
Scope of names
  definition, 204
Scope of statements, 203
Screen Section of Data Division, 97
SD entries, 95
SEARCH statement, 352-357
Section header
  definition, 28
SECURE clause, 169
  seg-key, 114
Segmentation, 245-246
Segmented key, defined, 3
SELECT optional in FILE-CONTROL paragraph, 61
Semicolon, as a separator, 16
Sentences, COBOL, 197
  compiler-directing, 202
  conditional, 202
  definition, 197
    imperative, 202
Separately compiled program
  definition, 1
Separators, 16-17
  definition, 16
Sequential files, 278, 393
Sequential search of a table, 353
Serial search of a table, 353
SET statement, 357-361
Sign
  default for unsigned operands, 226
    in arithmetic expressions, 221
    specifying position of, 170-171
    specifying representation of, 170-171
SIGN clause, 170-171
Sign condition, 229
  definition, 229
Sign control symbols, 155
  in fixed insertion editing, 155
    in floating insertion editing, 156
Simple condition
  definition, 224
Simple insertion editing, 155
SIN intrinsic function, 432
Size error condition
  description of, 235
    evaluation of exponentiation, 222
SORT statement, 362-368
  with table, 362
Sort-merge file description entries
  clauses of, 95
    structure of, 95
Sorting records, 362-368
  using the RELEASE statement, 346-347
  using the RETURN statement, 347-348
SOURCE clause, 171
Source program
  definition, 25
Source program reference format, 17
SPACE figurative constant, 14
Spaces, 16
  as delimiters
    arithmetic operators, 221
    of relational operators, 225
  as zero replacements, 118
    in file specification, 68
Special character word
  definition, 6
Special insertion editing, 155
Special registers, 3, 5, 6, 6-10, 10
  definition, 6
SPECIAL-NAMES paragraph, 14
Special-purpose word
  definition, 6
SQRT intrinsic function, 433
Standard Alignment Rules, 87
STANDARD-DEVIATION intrinsic function, 433
START statement, 369-373
Statements, COBOL, 197
  compiler-directing, 202
    conditional, 202
    definition, 197
    imperative, 202
    options of, 234-238
Status Key 1, 239
Status Key 2, 240
STOP statement, 373
Storage allocation, 88-93
  for COMP SYNC items, 86-93
  for group items, 83-93
  major-minor technique, 88-93
    when multiple entries describe the same area, 162-165
Storage format of a data item, 181-189
STRING statement, 374-378
Subscripting, 206-208
  basis for, 143
    in an identifier, 211
SUBTRACT statement, 378-380
Success/failure condition

510
<table>
<thead>
<tr>
<th>Definition</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM clause</td>
<td>172</td>
</tr>
<tr>
<td>SUM intrinsic function</td>
<td>434</td>
</tr>
<tr>
<td>SUPPRESS statement</td>
<td>380</td>
</tr>
<tr>
<td>SWITCH clause</td>
<td>47</td>
</tr>
<tr>
<td>Switch-status condition</td>
<td>228-229</td>
</tr>
<tr>
<td>definition</td>
<td>228</td>
</tr>
<tr>
<td>Switches</td>
<td></td>
</tr>
<tr>
<td>specifying in SPECIAL-NAMES paragraph</td>
<td>47</td>
</tr>
<tr>
<td>Symbolic-character</td>
<td>15</td>
</tr>
<tr>
<td>SYNCHRONIZED clause</td>
<td>175-176</td>
</tr>
<tr>
<td>Syntax rules</td>
<td></td>
</tr>
<tr>
<td>definition</td>
<td>23</td>
</tr>
<tr>
<td>purpose</td>
<td>23</td>
</tr>
<tr>
<td>System logicals</td>
<td>249</td>
</tr>
<tr>
<td>System-names</td>
<td>5-6</td>
</tr>
<tr>
<td>definition</td>
<td>5</td>
</tr>
<tr>
<td>Tab character</td>
<td>17</td>
</tr>
<tr>
<td>TAB key</td>
<td>use of in source file</td>
</tr>
<tr>
<td>Table handling</td>
<td></td>
</tr>
<tr>
<td>binary search for a table element</td>
<td>354</td>
</tr>
<tr>
<td>searching for a table element</td>
<td>352-357</td>
</tr>
<tr>
<td>sequential search for a table element</td>
<td>353</td>
</tr>
<tr>
<td>Table sort</td>
<td>362</td>
</tr>
<tr>
<td>Tables</td>
<td></td>
</tr>
<tr>
<td>defining (OCCURS clause)</td>
<td>143-146</td>
</tr>
<tr>
<td>rules for indexing</td>
<td>208-209</td>
</tr>
<tr>
<td>rules for subscripting</td>
<td>207-208</td>
</tr>
<tr>
<td>Tabs</td>
<td></td>
</tr>
<tr>
<td>in file specification</td>
<td>68</td>
</tr>
<tr>
<td>TAN intrinsic function</td>
<td>435</td>
</tr>
<tr>
<td>Technical note</td>
<td></td>
</tr>
<tr>
<td>definition</td>
<td>23</td>
</tr>
<tr>
<td>Terminal format</td>
<td></td>
</tr>
<tr>
<td>definition</td>
<td>22</td>
</tr>
<tr>
<td>TERMINATE statement</td>
<td>381</td>
</tr>
<tr>
<td>TEST-DATE-YYYYMMDD intrinsic function</td>
<td>435</td>
</tr>
<tr>
<td>TEST-DAY-YYYYYDDD intrinsic function</td>
<td>436</td>
</tr>
<tr>
<td>Testing</td>
<td></td>
</tr>
<tr>
<td>for the sign of a value</td>
<td>229</td>
</tr>
<tr>
<td>Text-word</td>
<td></td>
</tr>
<tr>
<td>definition</td>
<td>441</td>
</tr>
<tr>
<td>Transferring control in the Procedure Division</td>
<td>219-220</td>
</tr>
<tr>
<td>with CALL statement</td>
<td>272-276</td>
</tr>
<tr>
<td>with EXIT PROGRAM statement</td>
<td>302</td>
</tr>
<tr>
<td>with GO TO statement</td>
<td>304-305</td>
</tr>
<tr>
<td>with IF statement</td>
<td>306-308</td>
</tr>
<tr>
<td>with MERGE statement</td>
<td>316-319</td>
</tr>
<tr>
<td>with PERFORM statement</td>
<td>333-340</td>
</tr>
<tr>
<td>with READ statement</td>
<td>340-345</td>
</tr>
<tr>
<td>Truth value</td>
<td></td>
</tr>
<tr>
<td>definition</td>
<td>202</td>
</tr>
<tr>
<td>of conditional expressions</td>
<td>224</td>
</tr>
<tr>
<td>TYPE clause</td>
<td>176</td>
</tr>
<tr>
<td>U</td>
<td></td>
</tr>
<tr>
<td>Undefined results in a data-handling operation</td>
<td>238</td>
</tr>
<tr>
<td>UNDERLINE clause</td>
<td>180</td>
</tr>
<tr>
<td>Uniqueness of reference</td>
<td>3, 204</td>
</tr>
<tr>
<td>UNLOCK statement</td>
<td>382</td>
</tr>
<tr>
<td>UNSTRING statement</td>
<td>385-390</td>
</tr>
<tr>
<td>UPPER-CASE intrinsic function</td>
<td>437</td>
</tr>
<tr>
<td>USAGE clause</td>
<td>181-189</td>
</tr>
<tr>
<td>USAGE IS BINARY-CHAR clause</td>
<td>181</td>
</tr>
<tr>
<td>USAGE IS BINARY-DI auxiliary function ase</td>
<td></td>
</tr>
<tr>
<td>USAGE IS BINARY-LONG clause</td>
<td></td>
</tr>
<tr>
<td>USAGE IS BINARY-SHORT clause</td>
<td></td>
</tr>
<tr>
<td>USAGE IS FLOAT-EXTENDED clause</td>
<td></td>
</tr>
<tr>
<td>USAGE IS FLOAT-LONG clause</td>
<td></td>
</tr>
<tr>
<td>USAGE IS FLOAT-SHORT clause</td>
<td></td>
</tr>
<tr>
<td>USAGE IS POINTER-64</td>
<td></td>
</tr>
<tr>
<td>USE AFTER EXCEPTION procedure</td>
<td></td>
</tr>
<tr>
<td>and invalid key condition</td>
<td>244</td>
</tr>
<tr>
<td>USE statement</td>
<td>390-393</td>
</tr>
<tr>
<td>User-defined words</td>
<td>3-3</td>
</tr>
<tr>
<td>definition</td>
<td>3</td>
</tr>
<tr>
<td>uniqueness of</td>
<td>204</td>
</tr>
<tr>
<td>USING phrase</td>
<td>247</td>
</tr>
<tr>
<td>of CALL statement</td>
<td>273-275</td>
</tr>
<tr>
<td>of Procedure Division header</td>
<td>96, 273-</td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VALUE IS clause</td>
<td>189-195</td>
</tr>
<tr>
<td>use in Linkage Section</td>
<td>96</td>
</tr>
<tr>
<td>use in Working-Storage Section</td>
<td>96</td>
</tr>
<tr>
<td>VALUE OF ID clause</td>
<td>195</td>
</tr>
<tr>
<td>length limitation</td>
<td>195</td>
</tr>
<tr>
<td>Variable-length records</td>
<td>158-161</td>
</tr>
<tr>
<td>VARIANCE intrinsic function</td>
<td>438</td>
</tr>
<tr>
<td>Verbs, COBOL</td>
<td>197</td>
</tr>
<tr>
<td>definition</td>
<td>197</td>
</tr>
<tr>
<td>Version 2.8 – what's new since V2.5</td>
<td></td>
</tr>
<tr>
<td>/DISPLAY_FORMATTED, 286, 287</td>
<td></td>
</tr>
<tr>
<td>arithmetic, native and standard</td>
<td>221, 223</td>
</tr>
<tr>
<td>Arithmetic, standard</td>
<td>222</td>
</tr>
<tr>
<td>DISPLAY and [NOT] ON EXCEPTION</td>
<td>284</td>
</tr>
<tr>
<td>DISPLAY and END-DISPLAY</td>
<td>284</td>
</tr>
<tr>
<td>DISPLAY in an EXCEPTION for an ACCEPT</td>
<td>255, 292</td>
</tr>
<tr>
<td>DISPLAY, terminating (within ACCEPT ON EXCEPTION)</td>
<td>255</td>
</tr>
<tr>
<td>OPEN and APPLY PREALLOCATION to detect &quot;device full&quot;</td>
<td>332</td>
</tr>
<tr>
<td>OpenVMS I64 support, viii</td>
<td></td>
</tr>
<tr>
<td>status, passing to operating system (RETURN-</td>
<td></td>
</tr>
<tr>
<td>CODE and PROCEDURE DIVISION GIVING), 248</td>
<td></td>
</tr>
<tr>
<td>table sort, 362</td>
<td></td>
</tr>
<tr>
<td>WRITE and SYSS$FLUSH to detect &quot;device full&quot;, 398</td>
<td></td>
</tr>
</tbody>
</table>
Index

W
WHEN-COMPILED intrinsic function, 438
WINDOW phrase of the APPLY clause,
WITH IDENT clause, 33
   example, 31
Working-Storage Section of Data Division, 95
WRITE statement, 393-399
   effect on LINAGE values, -137

Y
Year 2000
   ACCEPT statement, 252
   CURRENT-DATE function, 410
YEAR-TO-YYYY intrinsic function, 439

Z
Zero suppression editing, 156