IBM Library Server Print Preview

DOCNUM = GC26-8070-00
DATETIME = 02/17/95 14:48:12
BLDVERS = 1.2
TITLE = COBOL/VSE Migration Guide
AUTHOR =

PATH = /home/webapps/epubs/htdocs/book

TITLE Title Page

IBM COBOL for VSE/ESA
Migration Guide
Release 1

Document Number GC26-8070-00

April 1995

NOTICES Notices

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" in topic FRONT_1.

EDITION Edition Notice

First Edition (April 1995)

This edition applies to Version 1 Release 1 of IBM COBOL for VSE/ESA, Program Number 5686-068, and to any subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, Department J58
P.O. Box 49023
San Jose, CA, 95161-9023
United States of America

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

CONTENTS Table of Contents

TABLES Tables

FIGURES Figures

FRONT_1 Notices

References in this publication to IBM products or services do not imply that they will be available everywhere IBM operates, nor that only IBM products or services can be used. Functionally equivalent products or services that do not infringe legal rights held by IBM can be used instead. Operation with products or services other than those expressly designated by IBM is your responsibility.

IBM may have patents or pending patent applications covering subject matter described herein. This document neither grants nor implies any license or immunity under any IBM or third-party patents, patent applications, trademarks, copyrights, or other similar rights, or any right to refer to IBM in any marketing activities. Other than responsibilities assumed via the Agreement for Purchase of IBM Machines and the Agreement for IBM Licensed Programs, IBM assumes no responsibility for any infringement of third-party rights that may result from use of the subject matter disclosed in this publication or from the manufacture, use, lease, or sale of machines or programs described herein.

Licenses under utility patents held by IBM are available on reasonable and nondiscriminatory terms. IBM does not grant licenses under its appearance design patents. Direct licensing inquiries in writing to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594, U.S.A..

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS

IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. **This disclaimer does not apply in the United Kingdom or elsewhere if inconsistent with local law.**

Subtopics:

FRONT_1.1 Programming Interface

This book is intended to help you write programs using *IBM COBOL for VSE/ESA* (COBOL/VSE). This *COBOL/VSE Migration Guide* documents General-Use Programming Interface and Associated Guidance Information provided by COBOL/VSE.

General-Use programming interfaces allow the customer to write programs that obtain the services of COBOL/VSE.

FRONT_1.2 Trademarks

The following terms, denoted by an asterisk (*) on their first occurrences in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

AD/Cycle	BookManager	C/370
CICS	CICS/ESA	CICS/VSE
COBOL/370	DFSORT	IBM
Language Environment	MVS/ESA	MVS/XA
Operating System/2	OS/2	S/370
System/370	Systems Application Architecture	SAA
SQL/DS	VSE/ESA	VM/ESA

FRONT_2 About This Book

This book provides information to help you to upgrade your source programs to IBM COBOL for VSE/ESA (COBOL/VSE) and to move your run time to IBM Language Environment for VSE/ESA (LE/VSE).

To aid in upgrading your source programs to the COBOL 85 Standard supported by COBOL/VSE, this book provides descriptions of the language differences between the COBOL 74 Standard and the COBOL 85 Standard. It also describes the IBM* conversion tools available to aid in converting your source programs to COBOL/VSE programs.

To aid in moving your run time to LE/VSE, this book provides information on how to run existing phases under LE/VSE, including link-edit requirements for support and recommended run-time options for compatible behavior.

For both types of conversion, source and run time, this book describes sample strategies and scenarios.

Subtopics:

FRONT_2.1 Acknowledgement

IBM would like to acknowledge the assistance of the GUIDE COBOL Migration Task Force in the preparation of the *OS/VS COBOL to VS COBOL II Migration Guide*. The task force provided ideas, experience-derived information, and perceptive comments on the subject of OS/VS COBOL to VS COBOL II conversion.

The information received from this previous conversion experience, as well as input from many experienced OS/VS COBOL and VS COBOL II IBM customers, aided in the development of the *VS COBOL II Migration Guide*. The *COBOL/VSE Migration Guide* is based upon the information in the *VS COBOL II Migration Guide*.

FRONT_2.2 Publications Provided with COBOL/VSE

Publications provided with the COBOL/VSE product include the following:

Figure 1. IBM COBOL for VSE/ESA Publications

Task	Publication	Order number
Evaluation and Planning	*General Information*	GC26-8068
	Migration Guide	GC26-8070
	Installation and Customization Guide	SC26-8071
Programming	*Programming Guide*	SC26-8072
	Language Reference	SC26-8073
	Reference Summary	SX26-3834
Diagnosis	*Diagnosis Guide*	SC26-8528
Warranty	*Licensed Program Specifications*	GC26-8069

General Information
Contains high-level information designed to help you evaluate the COBOL/VSE product. This book describes new compiler and language features, application development with LE/VSE, and product support for industry standards.

Migration Guide
Contains detailed migration and compatibility information for current users of DOS/VS COBOL and VS COBOL II who wish to

migrate to, or reuse existing applications on, COBOL/VSE. This book also describes several migration aids or tools to help you plan a migration path for your installation.

Installation and Customization Guide
Provides information you will need in order to install and customize the COBOL/VSE product. Detailed planning information includes:

- System and storage requirements for COBOL/VSE
- Information about changing compiler option defaults during installation
- Information for installing the product in shared storage

Programming Guide
Contains guidance information for writing and compiling application programs using COBOL/VSE, including information on the following topics:

- Programming using new product features, such as intrinsic functions
- Processing techniques for VSAM and SAM files
- Debugging techniques using compiler options and listings
- Nested programming techniques
- Subsystem considerations

Language Reference
Provides syntax and semantic information about the COBOL language as implemented by IBM, including rules for writing source programs, and descriptions of IBM language extensions. This book is meant to be used in conjunction with the *COBOL/VSE Programming Guide*, which provides programming task-oriented information.

Reference Summary
Contains a convenient summary of the COBOL/VSE language syntax--including new intrinsic functions--as well as syntax for compiler options, compiler-directing statements, and the COBOL/VSE reserved word list.

Diagnosis Guide
Provides instructions for diagnosing failures in the COBOL/VSE compiler product that are not caused by user error. This book will help you construct a keyword string that allows you or IBM Service to search the product failure database for previously documented problems and appropriate corrections.

Licensed Program Specifications
Contains a product description and product warranty information for the COBOL/VSE compiler.

FRONT_2.3 Language Environment for VSE/ESA Publications

Other publications useful for developing applications with COBOL/VSE include the following publications provided with the LE/VSE product:

Figure 2. IBM Language Environment for VSE/ESA Publications

Task	**Publication**	**Order number**
Evaluation and Planning	*Fact Sheet*	GC26-8062
	Concepts Guide	GC26-8063
	Installation and Customization Guide	SC26-8064
Programming	*Programming Guide*	SC26-8065
	Debugging Guide and Run-Time Messages	SC26-8066
	Reference Summary	SX26-3835
Diagnosis	*Diagnosis Guide*	SC26-8060
Warranty	*Licensed Program Specifications*	GC26-8061

Fact Sheet
Provides a brief overview and description of LE/VSE.

Concepts Guide
Provides a detailed overview of program models and intended architecture for LE/VSE, the common run-time environment.

Installation and Customization Guide
Contains information needed to plan for installing and customizing the LE/VSE product.

Programming Guide.
Provides detailed information on the following topics:

- Directions for linking and running programs that use LE/VSE services
- Information on storage management, run-time message handling, and condition handling models
- Callable services and run-time options, and how to use them
- Instructions for writing programs that use interlanguage communication (ILC)

This book also contains language-specific run-time information.

Debugging Guide and Run-Time Messages
Provides detailed information on debugging techniques and services. Provides a listing of run-time messages and their explanations, as well as abend codes.

Reference Summary
Contains a convenient summary of the IBM Language Environment for VSE/ESA.

Diagnosis Guide
Provides instructions for diagnosing failures in the LE/VSE product that are not caused by user error. This book will help you construct a keyword string that allows you or IBM Service to search the product failure database for previously documented problems and appropriate corrections.

Licensed Program Specifications
Contains a product description and warranty information.

1.0 Overview

Subtopics:

1.1 Chapter 1. Introducing COBOL/VSE and LE/VSE

This chapter provides an overview of COBOL/VSE and LE/VSE and familiarizes you with the terminology used throughout this book. This chapter includes information on the following:

- Product Relationships - COBOL/VSE and LE/VSE
- Comparison of the DOS/VS COBOL, VS COBOL II, and COBOL/VSE Compilers
- LE/VSE's Run-Time Support for DOS/VS COBOL and VS COBOL II Programs
- Advantages of COBOL/VSE and LE/VSE
- Obstacles to Upgrading to COBOL/VSE and LE/VSE
- Major Changes with COBOL/VSE and LE/VSE
- General Conversion Tasks

Subtopics:

1.1.1 Product Relationships - COBOL/VSE and LE/VSE

COBOL/VSE is IBM's strategic COBOL compiler for the VSE/ESA operating system. COBOL/VSE is a superset of DOS/VS COBOL and VS COBOL II, with

additional features such as intrinsic functions and support for a common run-time environment (LE/VSE). COBOL/VSE is based on IBM SAA* AD/Cycle* COBOL/370*.

LE/VSE provides a single run-time environment for all conforming high-level languages (currently COBOL/VSE and PL/I for VSE/ESA*). In addition to support for existing applications, LE/VSE also provides common conventions, common run-time facilities, and common callable services. These services offer benefits such as uniformity and consistency, improved interlanguage communication (ILC), reusable libraries, and simplified and more efficient application development. LE/VSE is required in order to run COBOL/VSE programs.

Figure 3 shows the relationship between previous IBM COBOL products and COBOL/VSE and LE/VSE.

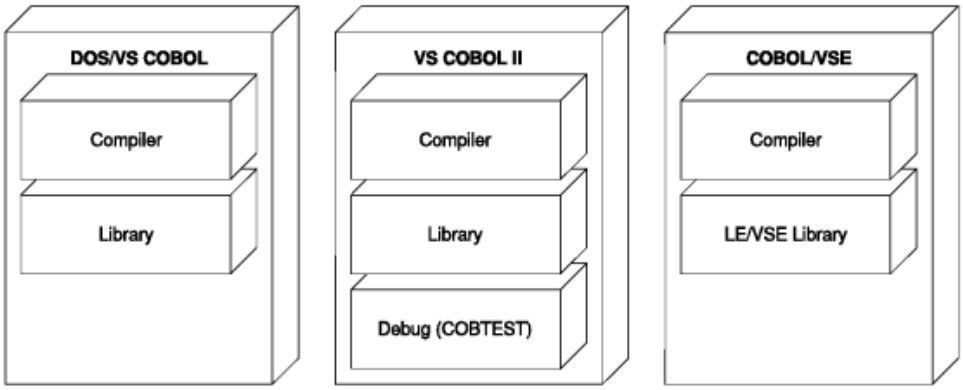


Figure 3. Product Relationships - COBOL/VSE and LE/VSE

1.1.2 VSE/ESA Version 1 Release 4 and VSE/ESA Version 2 Release 1 Support

If you intend to run COBOL/VSE under VSE/ESA Version 1 or later you must install COBOL/VSE Release 1 or later and LE/VSE Release 1 or later.

1.1.3 Comparison of IBM COBOLs

Figure 4 gives an overview of the functions available with the latest releases of DOS/VS COBOL and VS COBOL II, and shows the new function available with the COBOL/VSE compiler.

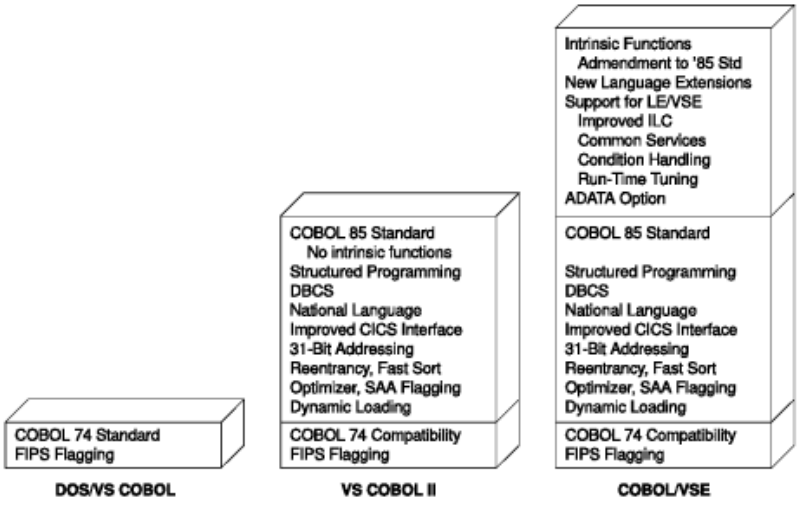


Figure 4. Comparison of IBM COBOLs

1.1.4 LE/VSE's Run-Time Support for DOS/VS COBOL and VS COBOL II Programs

LE/VSE provides compatibility support for programs that run under the DOS/VS COBOL and VS COBOL II libraries. LE/VSE also provides support for other LE/VSE-enabled language compilers, as shown in Figure 5.

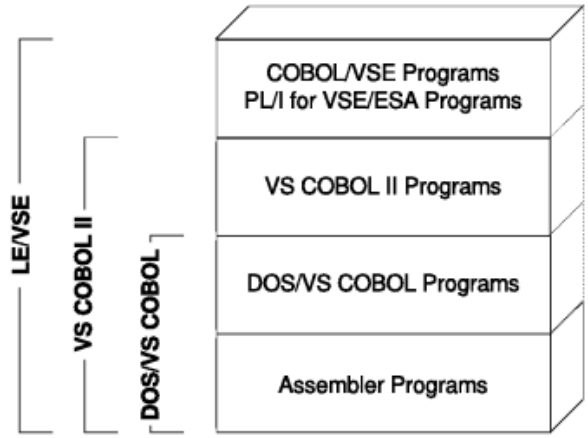


Figure 5. LE/VSE's Run-Time Support for DOS/VS COBOL and VS COBOL II Programs

---

1.1.5 Advantages of COBOL/VSE and LE/VSE

The COBOL/VSE compiler and LE/VSE run time provide additional functions

over DOS/VS COBOL and VS COBOL II. This section first describes the advantages of COBOL/VSE and LE/VSE over **both** VS COBOL II and DOS/VS COBOL, then the additional advantages COBOL/VSE and LE/VSE offer over DOS/VS COBOL only.

Subtopics:

1.1.5.1 Over VS COBOL II and DOS/VS COBOL

COBOL/VSE and LE/VSE provide the following benefits over both VS COBOL II and DOS/VS COBOL.

° The solution to the year 2000 problem facing many sites by using either:

- COBOL/VSE intrinsic functions
- LE/VSE callable services.

° The ability to have application-specific error-handling routines intercept program interrupts, abends, and other software-generated conditions for error recovery. This is done using COBOL/VSE procedure-pointer support and user-registered condition handlers.

° Cost savings by replacing multiple language run times with the single LE/VSE run time. Talk with your IBM representative to evaluate the potential cost savings based on the number of current licenses and languages used at your site.

° More flexible and comprehensive run-time options.

° Improved interlanguage communication (ILC) between COBOL and other languages, such as PL/I.

° Increased control over compiler outputs, such as Associated Data.

1.1.5.2 Over DOS/VS COBOL

COBOL/VSE and LE/VSE provide the following benefits over DOS/VS COBOL:

° Improved bit and character manipulation using Hex literals

° Improved flexibility with character manipulation using reference modification

° Virtual storage constraint relief (VSCR) to allow your programs to reside below or above the 16-megabyte line. Programs can also be compiled below or above the 16-megabyte line

° Top-down modular program development through nesting of programs, and improved CALL and COPY functions

° Straightforward structured program coding through:

- Inline PERFORM statements
- The CONTINUE place-holder statement
- The EVALUATE statement
- Explicit scope terminators (for example: END-IF, END-PERFORM, END-READ)

° Improved, easy-to-use program listings and maps

° COBOL 85 Standard conformance

° COBOL/VSE compiler options that give you added control over compiler output, such as:

- Object code generation
- Compiler usage of virtual storage
- Listings, maps, and diagnostics
- Run-time debugging information
- Delimiters for literals
- Customized reserved word lists
- Processing COPY or BASIS statements
- Text of error messages

° Improved data and control flow through use of the OPTIMIZE compiler option, which provides expanded code optimization

° Support for CICS*, DFSORT/VSE, SQL/DS*

° Support for reentrancy

1.1.6 Obstacles to Upgrading to COBOL/VSE and Moving to LE/VSE

Prerequisite products and function not yet available in LE/VSE might delay your upgrade to COBOL/VSE and LE/VSE. Figure 6 lists the obstacles that might prevent you from upgrading at this time.

Figure 6. Obstacles to Upgrading

Enterprise System Architecture (ESA)	VSE/ESA is required to run applications with LE/VSE. For a complete list of the ESA version and release levels supported, see the *LE/VSE Licensed Program Specifications*.
CICS/VSE Version 2 Release 3	LE/VSE Release 1 and later require CICS/VSE Version 2 Release 3.
Other High-Level Languages	Currently, LE/VSE supports COBOL/VSE and PL/I VSE/ESA. Interlanguage communication with

	other high-level languages might be limited or nonexistent with LE/VSE. (This does not apply to assembler language programs.)
Vendor Product Support	Any vendor products used by your site should be LE/VSE-enabled for the LE/VSE TRAP(ON) run-time option. Contact your vendors to verify that their products are designed to work with TRAP(ON).

If you find that any of the above prevents you from upgrading to COBOL/VSE and LE/VSE, you can still take incremental steps to prepare for conversion when these obstacles no longer apply. For example:

- Evaluate the effort to move to LE/VSE
- Code applications based on COBOL/VSE and LE/VSE requirements to ease a future conversion. For example, specify the RES compiler option instead of the NORES compiler option. (COBOL/VSE does not support the RES/NORES compiler option, and all programs are RES-like under LE/VSE.)
- Convert all source code to COBOL 85 Standard
- Upgrade to VS COBOL II Version 1 Release 4

1.1.7 Major Changes with COBOL/VSE and LE/VSE

With LE/VSE, you will find that existing applications are affected in two areas: abends after severe errors and the RES/NORES environment. A brief description of the differences and actions required to ensure compatibility follows:

Change Default LE/VSE Run-Time Options: The DOS/VS COBOL and VS COBOL II run-time options do not control whether or not your programs with severe errors end with an abend. With DOS/VS COBOL and VS COBOL II, all severe errors result in abends.

LE/VSE's default run-time option settings allow you to intercept and handle severe errors instead of ending the application with an abend. To ensure your application ends with an abend when there is a severe error, specify the LE/VSE ABTERMENC(ABEND) run-time option as an installation default.

Depending on your site's needs, you might need to change other run-time option settings supplied by IBM. For a list of recommended run-time option settings, see "Set Recommended Default LE/VSE Run-Time Options" in topic 3.1.1.

Understand the RES Environment: COBOL/VSE does not provide the RES/NORES compiler option. All programs are RES under COBOL/VSE. For existing applications compiled with NORES, two factors apply:

- If you recompile existing programs with your current compiler, specify the RES compiler option instead of the NORES compiler option. This allows you to use some LE/VSE services after link-editing with LE/VSE and also eliminates some link-edit requirements for a future move to LE/VSE. For more details, see Chapter 12, "Exploiting LE/VSE by Adding COBOL/VSE Programs to Existing Applications" in topic 5.1.

- Since programs compiled RES access the library when invoked, you now need to determine whether you place LE/VSE in the permanent LIBDEF chain or temporary LIBDEF chain, depending on your site's current configuration.

1.1.8 General Conversion Tasks

Depending on your site's situation, you will most likely need to complete one or more of the general conversion tasks, which include:

- Planning Your Strategy
- Moving Your Run Time to LE/VSE
- Upgrading Your Source to COBOL/VSE
- Adding COBOL/VSE to existing DOS/VS COBOL and VS COBOL II Applications

Each of these general conversion tasks have subtasks, which are described in detail throughout this book.

Subtopics:

1.1.8.1 Plan Your Strategy

Before upgrading your source programs to COBOL/VSE and moving your run time to LE/VSE, you should develop a conversion strategy. A thorough strategy should help ensure a smooth transition to COBOL/VSE and LE/VSE.

Your conversion plan might include staggering the upgrade to COBOL/VSE after the move to LE/VSE, depending on your site's needs. This book provides separate strategies for upgrading source and moving run time. Chapter 2, "Setting up a Strategy for Moving to the LE/VSE Run Time" in topic 2.1, describes one strategy for moving your run time. Chapter 3, "Setting up a Strategy to Upgrade Source to COBOL/VSE" in topic 2.2, describes a strategy for upgrading your source.

1.1.8.2 Move Your Run Time to LE/VSE

You can run existing phases with LE/VSE and receive the same results as with your current run-time library. In some cases, you will need to

link-edit the application with LE/VSE or compile the programs with COBOL/VSE. For details see:

- Chapter 5, "Moving from the DOS/VS COBOL Run Time to LE/VSE" in topic 3.2
- Chapter 6, "Moving from the VS COBOL II Run Time to LE/VSE" in topic 3.3

For details on actions required to ensure compatibility once you are ready to run your programs on LE/VSE, see Chapter 4, "Running Existing Applications under LE/VSE" in topic 3.1.

1.1.8.3 Upgrade Your Source to COBOL/VSE

The effort required to upgrade your source programs is dependent on the compiler used and the language level used.

VS COBOL II

From a conversion standpoint, no language differences exist between VS COBOL II Release 4 and COBOL/VSE. In fact, the only difference between VS COBOL II Release 3.2 and Release 4 are three minor ANSI interpretation changes. If your site currently is compiling VS COBOL II programs with the NOCMPR2 compiler option (and therefore coding to the COBOL 85 Standard), you can compile the programs with COBOL/VSE without change and receive the same results (depending on your use of the three elements affected by the ANSI interpretation changes). For details, see Chapter 9, "Modifying Your VS COBOL II Source Programs" in topic 4.3.

Programs compiled with CMPR2 are coded to the COBOL 74 Standard. Conversion is required in order for the programs to conform to the COBOL 85 Standard. Conversion aids can help with the effort to upgrade your source programs to COBOL/VSE. Details of language differences between CMPR2 and NOCMPR2 are included in the *VS COBOL II Migration Guide for VSE*.

Note: The only feature supported by VS COBOL II but not COBOL/VSE is the support for the associated file modes of the 3525 multi-function card device.

DOS/VS COBOL

DOS/VS COBOL programs compiled with LANGLVL(2) also normally are coded to the COBOL 74 Standard. Conversion is required in order for these programs to conform to the COBOL 85 Standard. DOS/VS COBOL programs compiled with LANGLVL(1) could contain additional statements that require conversion. For details, see Chapter 7, "Modifying Your DOS/VS COBOL Source Programs" in topic 4.1.

For details on the conversion aids available to upgrade source programs, see Appendix B, "Conversion Aids for Source Programs" in topic APPENDIX1.2.

1.1.8.4 Add COBOL/VSE to Existing DOS/VS COBOL and VS COBOL II Applications

You can create new COBOL/VSE applications (or recompile existing programs with COBOL/VSE) and run them with existing applications under LE/VSE.

When adding COBOL/VSE programs to existing applications, you need to be aware of link-edit requirements, and of restrictions applying to running programs above or below the 16-megabyte line. For details, see Chapter 12, "Exploiting LE/VSE by Adding COBOL/VSE Programs to Existing Applications" in topic 5.1.

2.0 Strategies

Subtopics:

2.1 Chapter 2. Setting up a Strategy for Moving to the LE/VSE Run Time

This chapter describes a general strategy for moving your run-time environment to LE/VSE. After performing a cost/benefit analysis and determining that moving to the LE/VSE run time is appropriate for your site, the following tasks are necessary, and should be performed in roughly the following order:

1. Prepare to move your run time to LE/VSE
2. Take an inventory of COBOL applications
3. Implement LE/VSE in test mode
4. Cut over to production use

Subtopics:

2.1.1 Prepare to Move your Run Time to LE/VSE

In preparing to move your run time to LE/VSE, you need to perform the following tasks, which can be done concurrently:

- Install LE/VSE
- Educate your programmers about LE/VSE
- Assess storage requirements

Subtopics:

2.1.1.1 Install LE/VSE

To install LE/VSE, see *LE/VSE Installation and Customization Guide*.

Important: To ensure the LE/VSE run-time results are compatible with your current run-time results, you will need to change the default run-time options. For a list of recommended run-time options, see "Set Recommended Default LE/VSE Run-Time Options" in topic 3.1.1.

2.1.1.2 Educate your Programmers about LE/VSE

You should ensure that your application programmers are familiar with the features of LE/VSE and the differences between your current run-time environment and the LE/VSE run time. Once your programmers are familiar with LE/VSE, they can then assist you in taking the inventory of applications as described in "Take an Inventory of COBOL Applications" in topic 2.1.2.

2.1.1.3 Assess Storage Requirements

Due to DOS/VS COBOL and VS COBOL II compatibility routines, new function, and support for other languages, storage requirements for LE/VSE are larger than your current COBOL library.

DASD Storage Requirements: During conversion you will need DASD storage for the LE/VSE run time as well as your current run-time library. When you have finished moving to the LE/VSE run-time environment, you will be able to free the storage reserved for the DOS/VS COBOL and/or the VS COBOL II run-time libraries.

To determine the amount of DASD storage required by LE/VSE, see *LE/VSE Installation and Customization Guide*.

Virtual Storage Requirements: Virtual storage requirements for placing library routines above or below the line have increased. During conversion, this might entail negotiations among programmers when storage below the 16-megabyte line is scarce and you are in the most intense stages of the conversion process.

Below-the-line storage requirements for batch applications increase by at least 1 megabyte, if using the default storage values supplied by IBM.

You can use the LE/VSE RPTSTG run-time option to generate a report of the actual storage used by your program or run unit. You can then adjust the storage values based on actual storage used. Or, for recommended values, see "Set Recommended Default LE/VSE Run-Time Options" in topic 3.1.1.

Above-the-line storage requirements can increase up to 72K with LE/VSE.

2.1.2 Take an Inventory of COBOL Applications

When taking an inventory of your COBOL applications, you need to gather information on the program attributes that require action, involve testing, and affect performance when moving to LE/VSE.

To determine how to use the information in this chapter and throughout the rest of this book, you need to know which programs are compiled with the DOS/VS COBOL compiler and which programs are compiled with the VS COBOL II compiler (this of course, is only a concern for sites using both DOS/VS COBOL and VS COBOL II).

Then, for your inventory, you should determine:

° For moving your programs to LE/VSE:

- Programs compiled with RES and NORES
- Run-time options used (and how specified)
- COBOL programs calling, or called by, assembler programs
- COBOL programs using interlanguage communication (PL/I)
- COBOL programs used under CICS, DL/I, SQL/DS, or other subsystems
- Control statements used
- Vendor products used and whether they work with LE/VSE
- Frequency and types of abends

° For regression testing:

- Test cases required and available

° For performance measurements:

- Amount of storage used
- Frequency of execution of reusable/common modules
- Program execution time (both CPU and elapsed)

Subtopics:

2.1.2.1 Define Complexity Ratings

Figure 7 defines complexity ratings for existing applications based on

three possible actions:

- Run under LE/VSE without change
- Run under LE/VSE after link-editing with LE/VSE
- Run under LE/VSE after changing source and link-editing with LE/VSE

Figure 7. Description of Complexity Ratings

Complexity Rating	Effort Needed
0	Runs under LE/VSE without change and without link-editing with LE/VSE
1 to 5	Requires moderate testing Requires moderate coordination Runs under LE/VSE without error if link-edited with LE/VSE
6 to 10	Requires moderate to high degree of coordination Requires moderate to high degree of testing Requires rewrite of module Requires link-edit with LE/VSE

2.1.2.2 Assign Complexity Ratings

Based on the range of complexity ratings shown in Figure 7 you can now assign a complexity rating to each phase within an application.

Use the highest complexity rating listed as the overall rating for that phase. For an application, the highest complexity rating you assign for any phase within the application is the complexity rating for the entire application.

Subtopics:

- 2.1.2.2.1 When Moving from DOS/VS COBOL Run Time
- 2.1.2.2.2 When Moving from the VS COBOL II Run Time

2.1.2.2.1 When Moving from DOS/VS COBOL Run Time

Figure 8 shows estimated complexity ratings for conversions of specific program attributes:

Figure 8. Complexity Ratings for DOS/VS COBOL Program Attributes

Program Attribute	Complexity Rating
Assembler programs issuing a STXIT AB/STXIT PC	10

Calls assembler routines using high-order byte of Register 13	10
Assembler programs are coded based on the internals of the ILBD routines	10
Assembler programs not following normal save area conventions	8
Link-editing programs using ILBDSET0 with the assembler driver	3
CICS (if using dynamic calls)	4
ILC (Interlanguage Communication) with PL/I programs	7

For details on mixed DOS/VS COBOL and assembler programs, see Appendix C, "Mixed COBOL and Assembler Applications" in topic APPENDIX1.3. For details on other DOS/VS COBOL program attributes, see Chapter 5, "Moving from the DOS/VS COBOL Run Time to LE/VSE" in topic 3.2.

2.1.2.2.2 When Moving from the VS COBOL II Run Time

Figure 9 shows estimated complexity ratings for conversions of specific program attributes.

Figure 9. Complexity Ratings for Programs Running under the VS COBOL II Run Time

Program Attribute	**Complexity Rating**
Compiled with NORES, multiple phases	3
Specifies MIXRES run-time option	5
Specifies MIXRES run-time option for DOS/VS COBOL programs without using IGZBRDGV	10
Use of IGZEOPT object module (for non-CICS applications)	2
Called by assembler routine with a LOAD and Branch	3
Using ILBDSET0 or IGZERRE with the assembler driver	3
Link-editing programs using ILBDSET0 or IGZERRE with the assembler driver when the assembler program LOADs these routines	0
Assembler programs issuing a STXIT AB/STXIT PC	10
Assembler programs not following normal save area conventions	8
ILC between VS COBOL II and PL/I programs	4
ILC between DOS/VS COBOL and PL/I programs	10
CICS online (if using dynamic calls)	4
Use of IGZETUN object module	2

For details on mixed VS COBOL II and assembler programs, see Appendix C, "Mixed COBOL and Assembler Applications" in topic APPENDIX1.3. For details on programs mixing RES and NORES applications, see Appendix D, "Mixed RES and NORES Applications" in topic APPENDIX1.4. For details on other VS COBOL II program attributes, see Chapter 6, "Moving from the VS COBOL II Run Time to LE/VSE" in topic 3.3.

2.1.3 Implement LE/VSE in a Test Environment

When you begin to implement LE/VSE, you should perform the following tasks. Once again, these tasks are very closely related and will most likely be done concurrently:

- Decide how to implement LE/VSE
- Set up a regression testing procedure
- Take performance measurements

Subtopics:

2.1.3.1 Decide How to Implement LE/VSE

The method you choose to move LE/VSE into production use is dependent on your site's configuration. Two general methods are possible: adding LE/VSE to the permanent LIBDEF chain or using a temporary LIBDEF chain approach.

Permanent LIBDEF Chain: Once you add LE/VSE to the permanent LIBDEF chain, LE/VSE is available to all applications referenced by the permanent LIBDEF chain. You should ensure all applications are functioning correctly under LE/VSE before adding LE/VSE to your permanent LIBDEF chain.

Temporary LIBDEF Chain: You can choose to implement LE/VSE gradually by using the temporary LIBDEF chain approach, which is the recommended method. When you select this approach, you implement one set of applications at a time. For example, if you have multiple processors linked together with channel-to-channel connections, you must treat the entire system as one processor and should convert subsystem by subsystem. For example, first convert all VSE subsystems and then the CICS subsystem. Convert batch applications last.

You will need to revise your JCL in order to apply the temporary LIBDEF chain to the LE/VSE run time during initial setup. You also might need to specify a CEEDUMP DLBL if you wish to direct LE/VSE dump output to a disk file.

Here is one example of how to implement LE/VSE using the temporary LIBDEF

chain method: for an organization that has a central development center (all compiling and linking is done in one location), and separate production sites:

1. Certify LE/VSE and COBOL/VSE at the central development center

   - Run tests with captured data on the DOS/VS COBOL run time and save all outputs.

   - Install LE/VSE in a temporary LIBDEF chain environment. This means that unchanged jobs will run with the DOS/VS COBOL run time, and that some users can use the LE/VSE run time by using temporary LIBDEF chain JCL to access the LE/VSE run-time library.

     Note: For DOS/VS COBOL and for VS COBOL II programs compiled with the NORES option, you must link-edit the applications with the LE/VSE library in order to test the LE/VSE run time. To prevent overlaying of the phase that contains the DOS/VS COBOL version of the application, you can target the phase produced by the linkage editor to a "test" sublibrary that is specified by the LIBDEF JCL statement during COBOL/VSE testing.

   - Run tests with captured data on the LE/VSE run time, using the temporary LIBDEF chain environment, and compare all outputs to the DOS/VS COBOL run time. Run parallel tests throughout the certification cycle to ensure that your applications produce the same results when run with LE/VSE as they did with the DOS/VS COBOL run time.

     Note: Do not compile with COBOL/VSE yet: you are still certifying the LE/VSE run time.

   - Finally, compile your COBOL/VSE test applications into a temporary LIBDEF chain environment, and rerun the certification tests, using a temporary LIBDEF chain to the LE/VSE run-time library.

2. Install LE/VSE into the SVA (Shared Virtual Area) on the central development center's system and test

   - Run parallel tests of the nonmigrated versions of your existing applications using temporary LIBDEF chain to access the DOS/VS COBOL run time.

   - Run all new applications in the LE/VSE run-time environment before releasing to production runs.

3. Install the LE/VSE run time at one production site

   - Continue to run parallel tests of the nonmigrated versions of your existing applications with the DOS/VS COBOL run time in the temporary LIBDEF chain environment.

   - Run the LE/VSE run time for one month at this production site

4. Install the LE/VSE run time at all production sites

    - Optional: Continue to run parallel tests of the nonmigrated versions of your existing applications with the DOS/VS COBOL run time in the temporary LIBDEF chain environment.

    - Run the LE/VSE run time for one month at all production sites.

    - After one month, delete the DOS/VS COBOL run-time library.

5. Compile all new or changed applications with COBOL/VSE

Try to move the largest units of work that you can. Moving entire applications or run units at once ensures that interactions between programs within an application or run unit can be tested.

2.1.3.2 Set up a Regression Testing Procedure

While moving your current run time to the LE/VSE run time, you should set up a procedure for regression testing. You should first move your applications to a test environment, and ensure you receive the expected results when running under LE/VSE. Regression testing will probably consume the bulk of the time spent on your move to LE/VSE.

During testing, run your existing applications in parallel on your current run time and under the LE/VSE run time. This allows you to verify that the results are the same.

Once the program runs correctly, test it against a variety of data:

1. Locally--each program separately
2. Globally--programs in a run unit in interaction with each other

In this way, you can exercise all the program processing features to help ensure that there are no unexpected execution differences.

Analyze program output and, if the results are not correct, use the LE/VSE dump services output to uncover any errors present and correct those errors.

Make any further changes you need and then rerun, and if necessary, continue to debug.

After you move your existing applications to production use under the LE/VSE run time, you might want to continue to run the applications in parallel as a backup for critical applications.

2.1.3.3 Take Performance Measurements

After you compare run-time performance between LE/VSE and your current run-time environment, and have identified which applications, if any, need performance improvements, you can investigate the methods available to tune your programs and improve performance. For example, you can modify storage values using the LE/VSE run-time options. For recommended storage values, see "Set Recommended Default LE/VSE Run-Time Options" in topic 3.1.1.

2.1.4 Cut Over to Production Mode

When your testing shows the entire application (or group of applications) in a CICS region is free of errors, you can move the entire unit over to production use. However, in case of unexpected errors, you should be prepared for instant recovery:

- Under VSE/ESA batch, run the old version as a substitute from the latest productivity checkpoint
- Under CICS return to the last commit point and then continue processing from that point using the unmigrated COBOL program. (For SQL/DS*, use an SQL ROLLBACK WORK statement.)
- For batch applications, use your site's backup and restore facilities to recover

After cutover, monitor the application to ensure it is working properly.

2.2 Chapter 3. Setting up a Strategy to Upgrade Source to COBOL/VSE

This chapter describes a general strategy for upgrading to COBOL/VSE. This chapter includes information on the following:

- Planning to upgrade your source to COBOL/VSE
- Making application program updates

Subtopics:

2.2.1 Planning to Upgrade Your Source to COBOL/VSE

After performing a cost/benefit analysis and determining that upgrading your source to COBOL/VSE is appropriate for your site, the following tasks are necessary:

1. Learn about the conversion aids
2. Install COBOL/VSE and conversion aids
3. Use conversion aids to analyze the effort to upgrade
4. Educate your programmers on COBOL/VSE features
5. Assess storage requirements
6. Take an inventory of COBOL applications
7. Prioritize your applications
8. Set up conversion/no-conversion categories
9. Set up a conversion procedure
10. Set up a regression testing procedure

Subtopics:

2.2.1.1 Learn About the Conversion Aids

If you use the available conversion aids, you will find that upgrading to COBOL/VSE can be less difficult than expected. The following conversion tools can assist in upgrading your source programs to COBOL/VSE.

- DOS/VS COBOL MIGR compiler option (for DOS/VS COBOL programs)
- COBOL/VSE CMPR2, FLAGMIG and NOCOMPILE compiler options
- COBOL and CICS Command Level Conversion Aid for VSE (CCCA)

Other conversion aids include:

- COBOL Structuring Facility (COBOL/SF) program product
- COBOL Report Writer Precompiler program offering

These conversion aids are fully described in Appendix B, "Conversion Aids for Source Programs" in topic APPENDIX1.2.

2.2.1.2 Install COBOL/VSE and Conversion Aids

If you have not already done so, you should now install COBOL/VSE. To install COBOL/VSE, see *COBOL/VSE Installation and Customization Guide*.

If you plan to use any of the conversion aids, you should install them at this time. For details, see the documentation for the conversion aid program(s) you plan to use.

2.2.1.3 Use Conversion Aids to Analyze the Effort to Upgrade

To analyze the size of the effort to upgrade your source programs, you can use the following conversion aids:

- DOS/VS COBOL MIGR compiler option (DOS/VS COBOL only)
- COBOL/VSE FLAGMIG and CMPR2 compiler options
- CCCA program offering

These conversion aids are described in Appendix B, "Conversion Aids for Source Programs" in topic APPENDIX1.2.

The DOS/VS COBOL MIGR option and the COBOL/VSE FLAGMIG and CMPR2 compiler options, provide you with a listing of source statements that might need conversion to COBOL/VSE NOCMPR2.

The CCCA program offering provides you with either a report of the statements that need to be changed or the actual converted program itself. To avoid statements from automatically being converted, use the option that suppresses the generation of source code.

2.2.1.4 Educate Your Programmers on COBOL/VSE Features

Early in the conversion effort, you should ensure that your application programmers are familiar with the features of COBOL/VSE and the relationship and interdependencies between COBOL/VSE and LE/VSE.

In addition to source language differences between the COBOL 68 Standard, COBOL 74 Standard, and COBOL 85 Standard, your programmers will need to be familiar with LE/VSE condition handling and LE/VSE callable services.

If your programmers are familiar with COBOL/VSE features, they can assist you in taking the inventory of programs as described in "Take an Inventory of COBOL Applications" in topic 2.2.1.6.

2.2.1.5 Assess Storage Requirements

You can load most of the COBOL/VSE compiler above the 16-megabyte line. In addition, COBOL/VSE object programs can run in 31-bit addressing mode and can reside above the 16-megabyte line. This frees storage below the 16-megabyte line, which you can use for programs and/or data that must reside below the line.

During conversion you will need DASD storage for your current COBOL compiler(s) as well as the COBOL/VSE compiler. When you have completed the conversion, and if you have upgraded all of your DOS/VS COBOL and/or VS COBOL II programs to COBOL/VSE, you will be able to free the storage reserved for your current COBOL compiler.

2.2.1.6 Take an Inventory of COBOL Applications

By taking an inventory of your COBOL applications, you get a detailed picture of the work that is required. (Be sure to include any third-party vendor applications you are using.) For each program, your inventory should include at least the following information:

VS COBOL II and DOS/VS COBOL:

- Programmer responsible
- COBOL Standard level of source program
- Compiler used (ANSI COBOL 68, DOS/VS COBOL, VS COBOL II)
- Compiler options used
- Precompiler options used
- Postprocessing options used
- COBOL modules
- COPY library members used in COBOL programs
- Called subprograms
- Calling programs
- Frequency of execution
- Test cases required and available

For DOS/VS COBOL only:

- Determine which programs contain Report Writer statements

- Determine which programs use the following features that are not supported by COBOL/VSE:

  - DAM files
  - Communications feature

- Determine which programs use features that might have different results under COBOL/VSE:

  - Variable-length data items (OCCURS DEPENDING ON)
  - Floating point numeric items
  - Exponentiation
  - Combined abbreviated relation conditions
  - Assembler routines using the high-order bit of Register 13

This information will be useful to you in the next step of your planning task, "Prioritizing Your Applications".

2.2.1.7 Prioritize Your Applications

Using the complete inventory, you can now prioritize the conversion effort. You should:

1. Define complexity ratings based on code conversion, testing, and coordination requirements

2. Assign complexity ratings to each item in your completed inventory and determine each program or application's resulting overall complexity rating

3. Determine the conversion priority of each program or application

Subtopics:

2.2.1.7.1 Define Complexity Ratings

Complexity ratings are defined based on the effort required to convert, test, and coordinate a construct or program. Figure 10 shows one example of how you might define complexity ratings. The ratings shown here are used in Figure 11 in topic 2.2.1.7.2.

Figure 10. Complexity Ratings--Conversion Effort Needed

Complexity Rating	Conversion Effort Needed
0	All code converted by CCCA without error; code compiles correctly under COBOL/VSE
1 to 3	Requires moderate testing Requires moderate coordination Most code converted without error by CCCA
4	Requires CCCA and possible manual conversion Requires special testing considerations
5 to 6	Requires moderate to high degree of coordination Requires moderate to high degree of testing for functional equivalence Requires conversion in addition to CCCA (manual or automated)
7 to 8	Requires high degree of coordination Requires high degree of testing for functional equivalence
9	Requires very high degree of coordination Requires very high degree of testing for functional equivalence
10	Requires rewrite of module

2.2.1.7.2 Assign Complexity Ratings

Based on the complexity ratings shown in Figure 10 in topic 2.2.1.7.1 (or your own defined complexity ratings), you can now assign a complexity rating to each attribute within a program.

Use the highest complexity rating listed as the overall rating for that program. For an application, the highest complexity rating you assign for any program within the application is the complexity rating for the entire application.

Figure 11 shows estimated complexity ratings for conversions of specific program attributes. On categories marked **M** you can gather information using the DOS/VS COBOL MIGR option. On categories marked **C** you can gather information using the CCCA program offering. You can also use the COBOL/VSE FLAGMIG compiler option to identify elements that need to be changed.

Figure 11. Complexity Ratings for DOS/VS COBOL Program Attributes

Program Attribute	Description of Attribute	Complexity Rating
Lines of Source Code	1000 or less	0
	5000 to 10,000	3
	10,000 to 20,000 +	5
COBOL 74 Standard COPY Library Members		1 M C
ANSI COBOL 68 COPY Library Members	1 to 10	2 M C
	10 to 20	5 M C
	20 +	6 M C
Stability	Program with no plans for changes	0
	Program changes twice a year	3
	Program changes every month or more often	8+
Files Accessed	1 to 3	1 M C
	3 to 5	3 M C
	6 +	6 M C
No Source Code for Module	Module needs rewrite(3)	10
	Module does not need to be upgraded	6
Compiled by Full ANSI COBOL 68 Compiler (pre-DOS/VS COBOL compiler)		4 C(1)
Compiled by DOS/VS COBOL R3.0 or R3.1 Compiler	LANGLVL(2) no manual changes	1 M C
	LANGLVL(1) no manual changes	1 M C
	LANGLVL(2) manual changes	4 M C
	LANGLVL(1) manual changes	4 M C
Access Methods Used	ISAM	6 M C
	DAM	10 C(2)
Uses Report Writer Language (if not using Report Writer Precompiler)		6 M C
Uses Report Writer Language (if using Report Writer Precompiler)		0
Uses Language with Changed Results	Complex OCCURS DEPENDING ON	4 C
	Combined abbreviated	6 M

	relation conditions		
	Floating-point	6	M
	arithmetic	6	M
	Exponentiation	2	
	Signed data	2	
	Binary data		
CICS		4	

Notes

1 In this table and in the sections that follow, any reference to Full ANSI COBOL 68 is a reference to IBM Full American National COBOL, Version 4 (Program 5736-CB2), or to LANGLVL(1) of DOS/VS COBOL (Program 5746-CB1).

2 This is a partial conversion.

3 Recreating source code can be done through non-IBM vendors.

2.2.1.7.3 Determine Conversion Priority

After you have determined the complexity rating for each program in your inventory, you can make informed decisions about the programs you want to upgrade, and the order in which you want to upgrade them.

Figure 12 shows one method of relating program complexity ratings to conversion priorities. (The highest priority is "1" and the lowest priority is "6".)

Figure 12. Assigning Program Conversion Priorities

Conversion Priority	**Complexity Rating**	**Other Considerations**
1	0 to 3	Great importance to organization Low conversion effort using conversion aids
2	4 to 6	Great importance to organization Medium conversion effort using conversion aids
	0 to 3	Medium importance to organization Low conversion effort using conversion aids
3	7 to 8	Great importance to organization High conversion effort using conversion aids
	3 to 6	Medium importance to organization Medium conversion effort using conversion aids
	0 to 3	Small importance to organization Low conversion effort using conversion aids
4	9 to 10	Great importance to organization Very high conversion effort
	7 to 8	Medium importance to organization High conversion effort using conversion aids
	3 to 6	Small importance to organization Medium conversion effort using conversion aids
5	9 to 10	Medium importance to organization Very high conversion effort
	7 to 8	Small importance to organization

		High conversion effort using conversion aids
6	9 to 10	Small importance to organization Very high conversion effort

You should also consider the following when deciding on conversion priorities:

- If your application is at the limits of System/370* architecture, it is a prime candidate for conversion to COBOL/VSE. With the VSE/ESA architecture you can obtain Virtual Storage Constraint Relief
- If the program does not run properly with LE/VSE, you must convert it

The priorities you establish together with the conversion effort required, can help you decide the order in which you want to convert your applications and programs.

2.2.1.8 Set Up Conversion/No-Conversion Categories

By using the conversion priorities you have established, and taking into account program importance and frequency of execution, you can list most of your programs in the order you want to convert them to COBOL/VSE.

There might be some programs that you do not want to convert at all, such as:

- Programs for which you have no source code, that will never need recompilation, and that run correctly with LE/VSE
- Programs of low importance to your organization that run correctly with LE/VSE and that would take a very high conversion effort

Note, however, that there might be restrictions on running programs with the converted modules. See Chapter 4, "Running Existing Applications under LE/VSE" in topic 3.1.

2.2.1.9 Set Up a Conversion Procedure

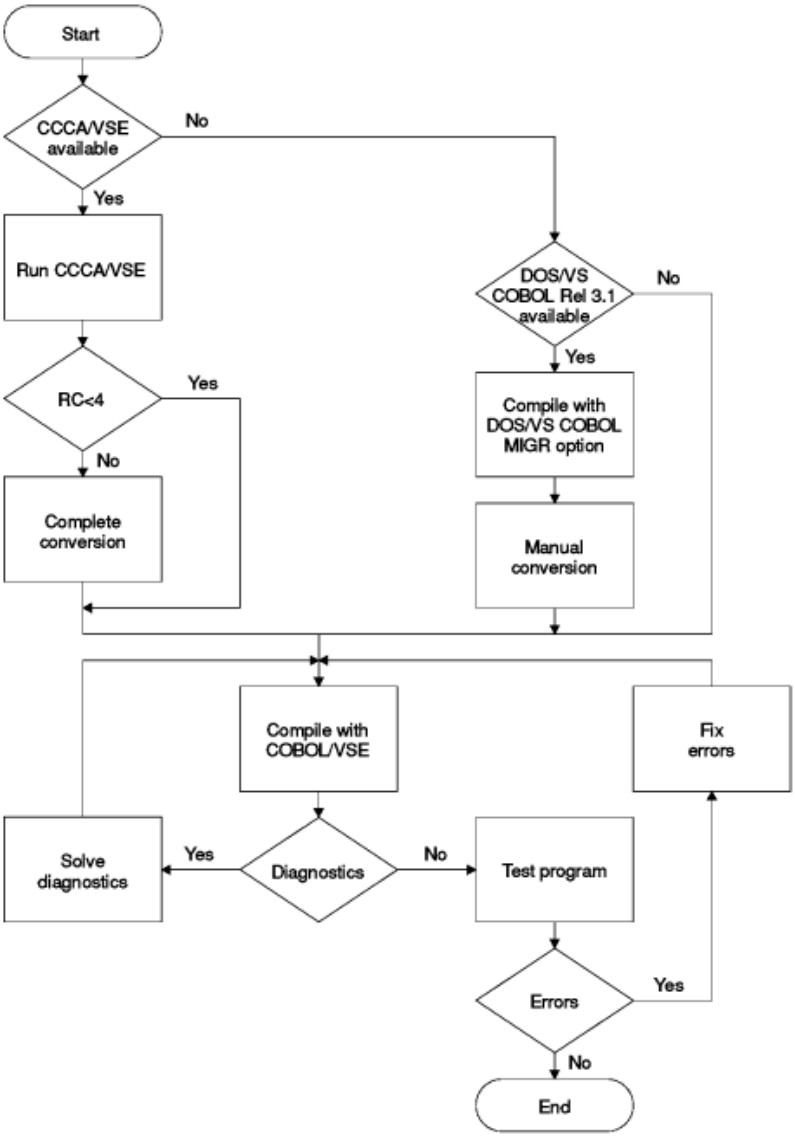
The summaries and diagrams on the following pages outline the steps required to upgrade five types of applications:

- Program without CICS or Report Writer
- Program converted to structured programming code
- Program with CICS
- Program with Report Writer statements to be discarded
- Program with Report Writer statements to be retained

Subtopics:

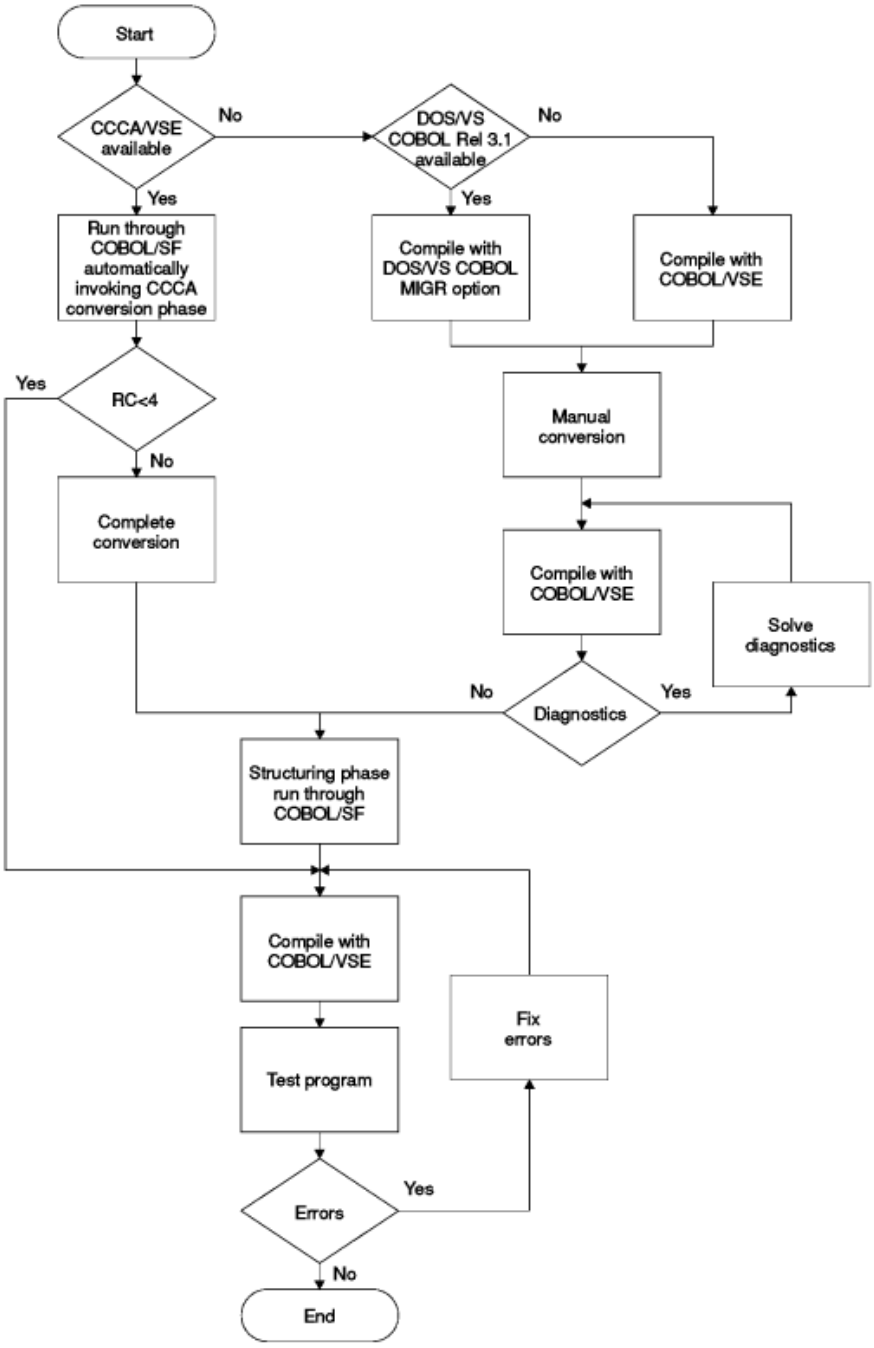
2.2.1.9.1 Program without CICS or Report Writer

You need to convert a DOS/VS COBOL program that contains neither CICS commands nor Report Writer statements to a COBOL/VSE program.



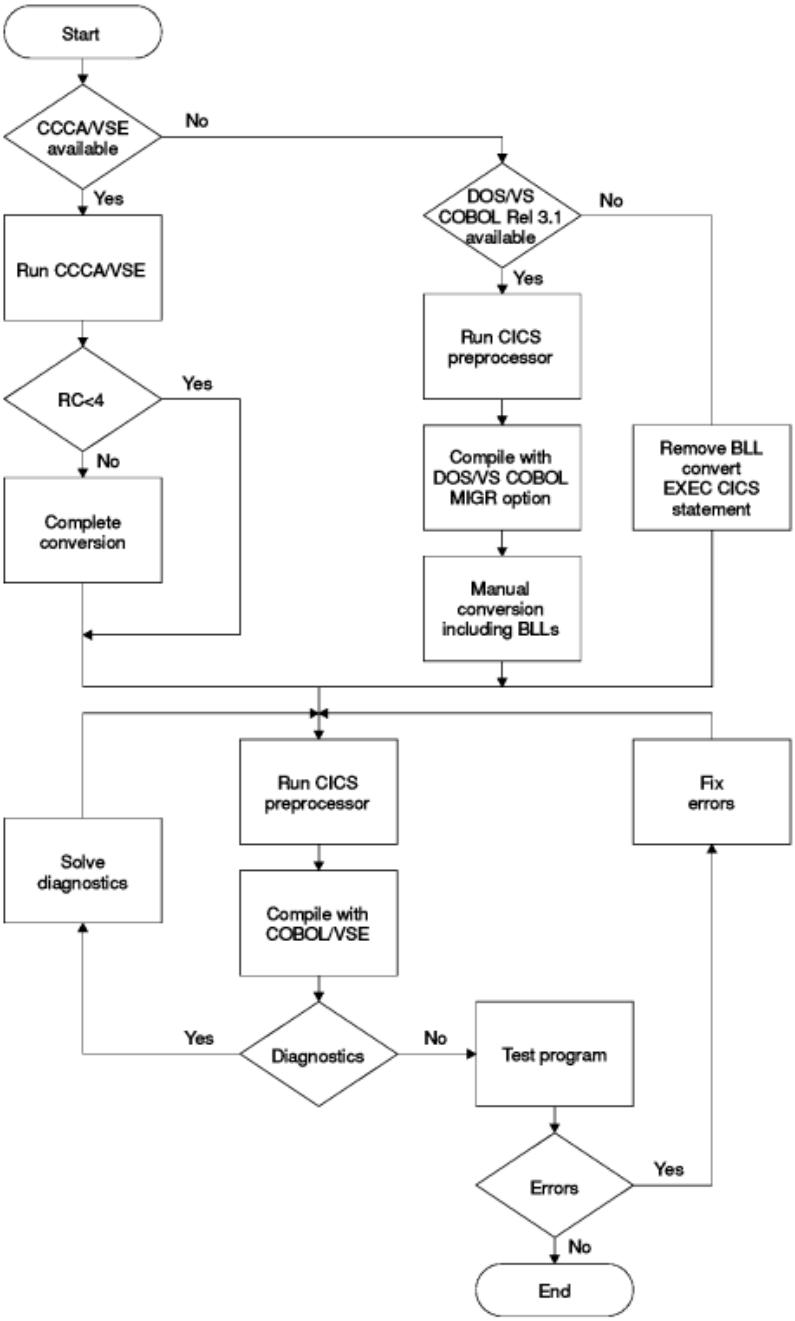
2.2.1.9.2 Program Converted to Structured Programming Code

You need to convert a non-CICS DOS/VS COBOL program, which does not contain Report Writer statements, to COBOL/VSE. Then, to convert to a structured format (this requires the availability of COBOL/SF) you do the following:



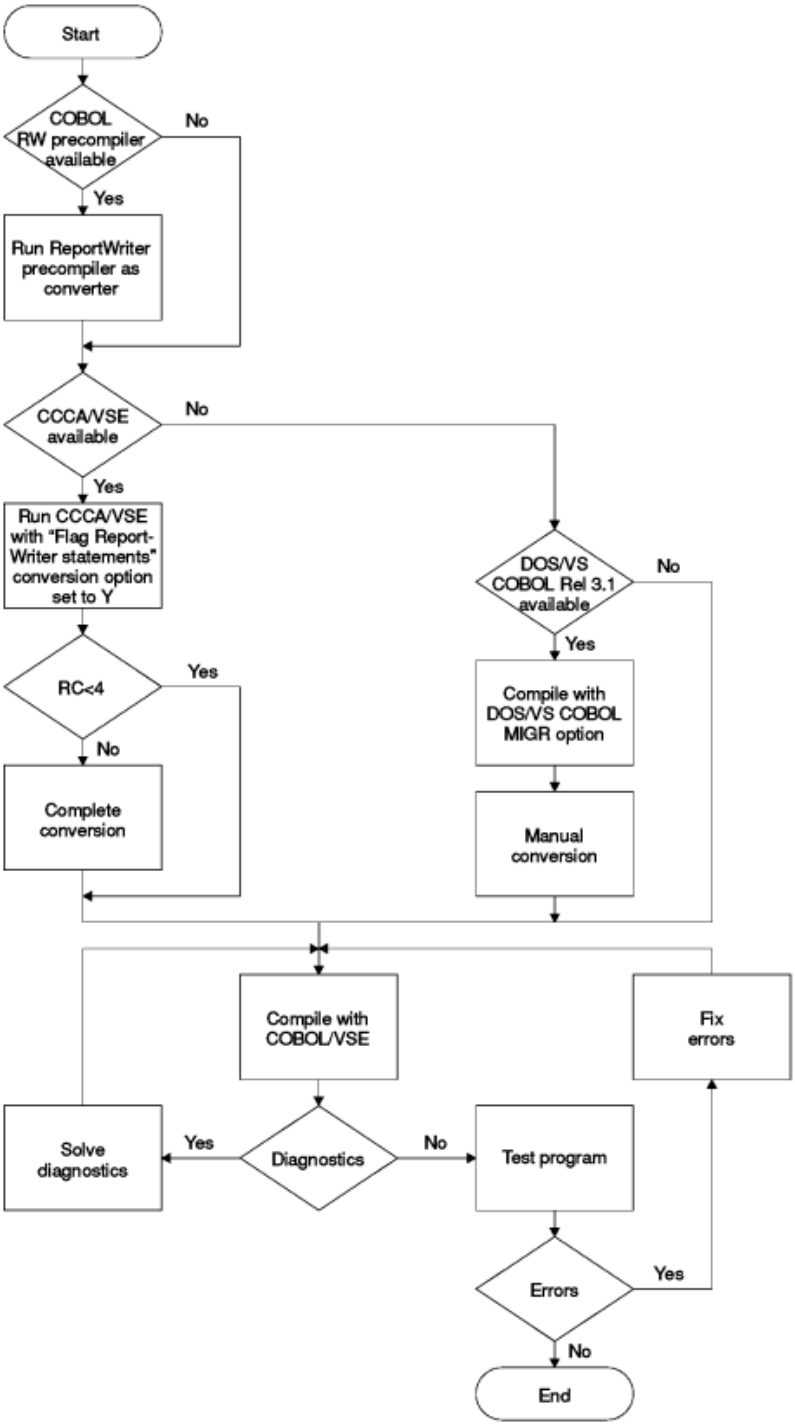
2.2.1.9.3 Program with CICS

You need to convert a DOS/VS COBOL program that contains CICS commands to a COBOL/VSE program.



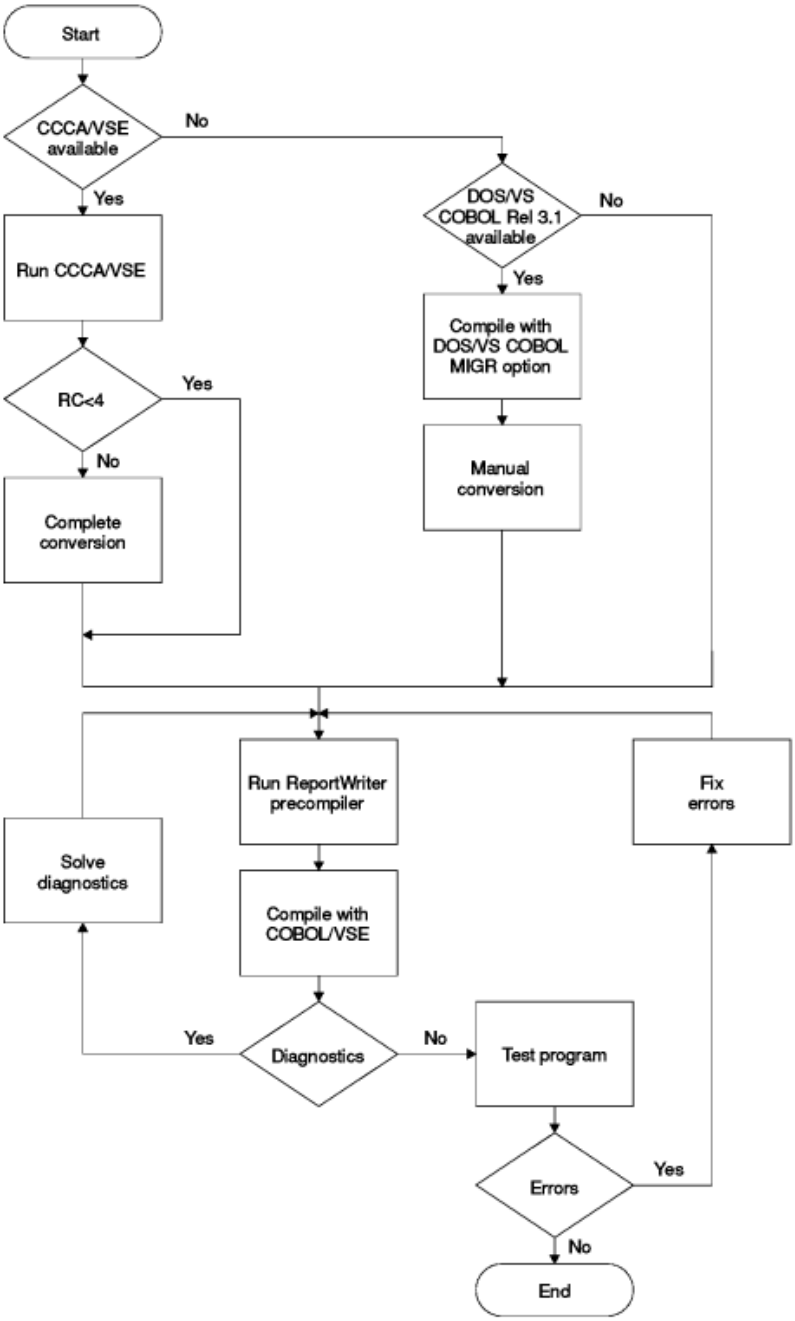
2.2.1.9.4 Program with Report Writer Statements to Be Discarded

```
You need to convert a DOS/VS COBOL program containing Report Writer
instructions to  COBOL/VSE.  You also remove all Report Writer statements.
```



2.2.1.9.5 Program with Report Writer Statements to Be Retained

```
You need to convert a DOS/VS COBOL program that contains Report Writer
instructions to a COBOL/VSE program.  You also decide to retain all the
Report Writer statements in the source code.
```



2.2.1.10 Set up a Regression Testing Procedure

After upgrading your source to COBOL/VSE, you should set up a procedure for regression testing. This should help to identify:

- File Status 39 for fixed file attributes mismatches. You should verify that your COBOL records match the JCL. For more information, see Appendix J, "Preventing File Status 39 for SAM Files" in topic APPENDIX1.10

- Working storage dependencies on zero values. You should specify the LE/VSE STORAGE(00) run-time option

---

2.2.2 Making Application Program Updates

The following application programming tasks are necessary when upgrading your source. They should be performed in roughly the following order:

Update Job and Module Documentation: It is extremely important that all updates be properly documented. COBOL itself is reasonably self-documenting. However, you should keep a log of the compiler options you specify and the reasons for specifying them. You should also document any special system considerations. This is an iterative process and should be performed throughout the conversion programming task.

Obtain Virtual and DASD Storage: During conversion, you will need extra virtual storage and extra DASD space. This might entail negotiations among programmers when storage below the 16-megabyte line is scarce and you are in the most intense stages of the conversion process.

Update Available Source Code: Whenever possible, use the conversion aids described in Appendix B, "Conversion Aids for Source Programs" in topic APPENDIX1.2. Otherwise, update the source code manually.

Revise JCL and Control Statements: By revising the JCL and control statements of your source programs, you can take advantage of the ways that COBOL/VSE supports VSE/ESA.

Compile, Link-Edit, and Run: Once the source is updated, you can process the program as you would a newly-written COBOL/VSE program. (Note: you need the LE/VSE run time installed.)

Debug: You can then analyze program output and, if the results are not correct, use the LE/VSE dump services output to uncover any errors present.

Repeat When Necessary: Make the corrections you need, and then recompile, relink, rerun, and, if necessary, continue to debug.

Test the Converted Programs: Once the program runs correctly, test it against a variety of data:

1. Locally--each program separately
2. Globally--programs in a run unit in interaction with each other

In this way, you can exercise all the program processing features to help ensure that there are no unexpected execution differences.

Repeat When Necessary: Make any further corrections you need, and then recompile, relink, rerun, and, if necessary, continue to debug.

Cut Over to Production Mode: When your testing shows the entire application is free of errors, you can move the entire unit over to production use. (This assumes your production system is already using the LE/VSE run time. If not, use the LIBDEF statement to specify the LE/VSE run time. See "Decide How to Implement LE/VSE" in topic 2.1.3.1.)

In case of unexpected errors, you should be prepared for instant recovery:

- Under VSE/ESA batch, run the old version as a substitute from the latest productivity checkpoint
- Under SQL/DS, CICS, and DL/I return to the last commit point and then continue processing from that point using the unmigrated COBOL program. (For SQL/DS, use an SQL ROLLBACK WORK statement.)
- For batch applications, use your site's backup and restore facilities to recover

Run in Production Mode: After cutover monitor the application to ensure you are getting the results expected. After that, your source conversion task is completed.

3.0 Moving Existing Applications to LE/VSE

Subtopics:

3.1 Chapter 4. Running Existing Applications under LE/VSE

In order to ensure that your current applications run under LE/VSE and provide the expected results, you need to know how to do the following:

- Set recommended default LE/VSE run-time options
- Invoke existing applications
- Link-edit existing applications
- Specify an abnormal termination exit to obtain a system dump

Other factors also apply in order to ensure compatibility. In addition to the information in this chapter, you should read the information in Chapter 5, "Moving from the DOS/VS COBOL Run Time to LE/VSE" in topic 3.2 and Chapter 6, "Moving from the VS COBOL II Run Time to LE/VSE" in topic 3.3, depending on whether you are moving your run time from DOS/VS COBOL or VS COBOL II.

Subtopics:

3.1.1 Set Recommended Default LE/VSE Run-Time Options

Depending on the characteristics of your applications, the LE/VSE default run-time option settings supplied by IBM might not provide the same run-time behavior as with the DOS/VS COBOL or VS COBOL II run-time libraries.

This section has two purposes. First, to inform you of the recommended run-time option settings for COBOL programs, so that you can determine which run-time options settings you need to change. Second, to ensure that you do not inadvertently change any default settings that are highly recommended for COBOL programs.

In some cases, the default run-time option settings supplied by IBM for applications run under CICS are different than the default settings for non-CICS applications. This section is divided into two parts: one part for non-CICS applications and one part for CICS applications. Each part lists the recommended default settings and the default supplied by IBM.

Subtopics:

3.1.1.1 Recommended Default Run-Time Options for Non-CICS Applications

Figure 13 describes the LE/VSE run-time options that are recommended for existing COBOL applications that do not run under CICS. For a complete list of LE/VSE run-time options, see Appendix E, "Run-time Option Comparison" in topic APPENDIX1.5.

Figure 13. Recommended LE/VSE Run-Time Options for COBOL Applications not Running Under CICS

Option	IBM Default Setting	Recommended Setting
ABTERMENC	RETCODE	ABEND

CBLOPTS	ON	ON
Storage Management		
ANYHEAP	(32K,16K,ANYWHERE,FREE)	(16K,8K,ANYWHERE,FREE)
BELOWHEAP	(32K,16K,FREE)	(8K,4K,FREE)
HEAP	(64K,64K,ANYWHERE,KEEP, 16K,16K)	(32K,32K,ANYWHERE,KEEP, 8K,4K)
LIBSTACK	(32K,16K,FREE)	(16K,8K,FREE)
STACK	(512K,512K,BELOW,KEEP)	(64K,64K,BELOW,KEEP)
TRAP	ON	ON

Note:

- ABTERMENC(ABEND) ensures you receive abend codes similar to those issued by VS COBOL II or DOS/VS COBOL when an abend, program check or severe error occurs

- CBLOPTS(ON) allows the existing COBOL format of the invocation character string to continue working (user parameters followed by the run-time options). This option is only valid for applications with COBOL as the main program

- For STACK, you must specify the suboption BELOW if any program in your application is running in AMODE 24. Otherwise, specify ANYWHERE.

- TRAP specifies how LE/VSE routines handle abends and program interrupts. In order for applications to run successfully, you must specify TRAP(ON)

  TRAP(ON) also enables LE/VSE to support the existing condition handling mechanisms provided by both VS COBOL II (STAE run-time option) and DOS/VS COBOL (STATE, FLOW, COUNT, and SYMDMP debugging options)

Subtopics:

- 3.1.1.1.1 Other Run-Time Options Affecting Non-CICS Programs

---

3.1.1.1.1 Other Run-Time Options Affecting Non-CICS Programs

In addition to 'normal' COBOL behavior (which you will get by using the recommended defaults listed above), some of your programs might have dependencies on other LE/VSE run-time options.

ALL31

ALL31(OFF) is required for applications with AMODE(24) programs, such as DOS/VS COBOL programs, VS COBOL II NORES programs, and other AMODE(24) non-COBOL programs.

ALL31(ON) allows external data to be allocated anywhere within the 31-bit addressing range and eliminates the need for AMODE switching.

The default supplied by IBM is ALL31(OFF).

STORAGE

For programs that depend on working storage that is initialized

to binary zeros, this option initializes any storage acquired by a reentrant program to binary zeros, except when VALUE clauses are specified. It also sets all the external data records of a program to binary zeros.

STORAGE(00,NONE,NONE,8K) is the LE/VSE equivalent of the VS COBOL II WSCLEAR run-time option.

The default supplied by IBM is STORAGE(NONE,NONE,NONE,8K).

3.1.1.2 Recommended Default Run-Time Options for CICS Applications

Figure 14 describes the LE/VSE run-time options that are highly recommended for existing COBOL applications that run under CICS. Under CICS, the default setting supplied by IBM is the same as the recommended setting. Therefore, no action is required. This information serves to ensure that you do not inadvertently reset any of the options to settings that do not provide compatible COBOL behavior. For a complete list of LE/VSE run-time options, see Appendix E, "Run-time Option Comparison" in topic APPENDIX1.5.

Figure 14. Recommended LE/VSE Run-Time Options for COBOL Applications Run Under CICS

Option	**IBM Default Setting and Recommended Setting**	**Comments**
ABTERMENC	ABEND	Ensures you receive abend codes similar to those issued by VS COBOL II or DOS/VS COBOL when an abend, program check, or severe error occurs.
ALL31	ON	ALL31(ON) allows LE/VSE to allocate its control blocks above the line. With ALL31(OFF), LE/VSE increases its use of below-the-line storage, which may cause CICS regions to go short on storage (SOS). ALL31(ON) can (and should) be used with all programs running under CICS, including DOS/VS COBOL programs.
CBLOPTS	ON	For applications with COBOL as the main program, CBLOPTS(ON) allows the existing COBOL format of the invocation character string to continue working (user parameters followed by the run-time options).
Storage Management ANYHEAP BELOWHEAP HEAP LIBSTACK	 (4K,4K,ANY,FREE) (4K,4K,FREE) (4K,4K,ANY,KEEP,4K,4K) (4K,4K,FREE)	LE/VSE provides these run-time options to help manage storage.

STACK	(4K,4K,ANY,KEEP)	
TRAP	ON	TRAP specifies how LE/VSE routines handle abends and program interrupts. In order for applications to run successfully, you must specify TRAP(ON).

Subtopics:

- 3.1.1.2.1 Other Run-Time Options Affecting CICS Programs

3.1.1.2.1 Other Run-Time Options Affecting CICS Programs

In addition to 'normal' COBOL behavior (which you will get by using the recommended defaults listed above), some of your programs might have dependencies on other LE/VSE run-time options.

CBLPSHPOP

CBLPSHPOP is used to control whether CICS PUSH HANDLE and CICS POP HANDLE commands are issued when a VS COBOL II or COBOL/VSE subroutine is called. CBLPSHPOP(ON) ensures you receive behavior that is compatible with VS COBOL II programs. CBLPSHPOP(OFF) can offer performance benefits. For details, see "CICS Language Elements that Suspend/Restore CICS Commands" in topic 4.5.1.3.

STORAGE

For programs that depend on working storage that is initialized to binary zeros, this option initializes any storage acquired by a reentrant program to binary zeros, except when VALUE clauses are specified. It also sets all the external data records of a program to binary zeros.

STORAGE(00,NONE,NONE,0K) is the LE/VSE equivalent of the VS COBOL II WSCLEAR run-time option.

The default setting supplied by IBM is STORAGE(NONE,NONE,NONE,0K).

3.1.2 Invoke Existing Applications

To access LE/VSE you will need to change the procedures you use for invoking applications. The procedures required for non-CICS applications are different than the procedures for CICS applications.

Subtopics:

- 3.1.2.1 For Non-CICS Applications
- 3.1.2.2 For CICS Applications

3.1.2.1 For Non-CICS Applications

The following information detail the changes required for non-CICS applications. For more information on how to prepare and run your programs with LE/VSE, see the *LE/VSE Programming Guide*.

Note: Make sure your program names do not begin with CEE or IGZ. These prefixes are reserved for LE/VSE library routine module names.

Specify the Correct Sublibrary: To invoke existing applications when running under LE/VSE, you need to replace the current 'sublibrary' with PRD2.SCEEBASE.

Specify Alternate FILENAMES (Optional): With LE/VSE, you can indicate the destination for LE/VSE output by changing the file name in the MSGFILE run-time option to the file names you want. Figure 15 lists the default file names for LE/VSE output.

Figure 15. Specification of New FILENAMEs

Output	**Default file name**
Messages	SYSLST
Run-time options report (RPTOPTS)	SYSLST
Storage reports (RPTSTG)	SYSLST
Dumps	CEEDUMP

3.1.2.2 For CICS Applications

For details on how to invoke COBOL applications run under CICS on LE/VSE, see *LE/VSE Installation and Customization Guide*.

Under CICS, LE/VSE output goes to a transient data queue named CESE. Each record written to the file has a header that includes the terminal-ID, the transaction-ID, date, and time. Figure 16 lists the types of LE/VSE output and location.

Figure 16. Location of LE/VSE Output under CICS

Output	**Transient Data Queue**
Messages	CESE
Run-time options report (RPTOPTS)	CESE
Storage reports (RPTSTG)	CESE
Dumps	CESE

3.1.3 Link-Edit Existing Applications

After determining which of your existing applications either require, or will benefit from, link-editing with LE/VSE, you need to specify the correct sublibrary name (PRD2.SCEEBASE).

To determine which applications require link-edit with LE/VSE, see Chapter 5, "Moving from the DOS/VS COBOL Run Time to LE/VSE" in topic 3.2 or Chapter 6, "Moving from the VS COBOL II Run Time to LE/VSE" in topic 3.3, depending on whether you are using the DOS/VS COBOL library or VS COBOL II library.

3.1.4 Specify Abnormal Termination Exit for System Dump

To receive a system dump under LE/VSE, you must specify an abnormal termination exit. With LE/VSE, you can invoke an abnormal termination exit before LE/VSE terminates a thread due to an unhandled condition of severity 2 or greater. Therefore, you can collect problem determination data before LE/VSE frees the resources it has acquired.

LE/VSE provides source for sample abnormal termination exits for both CICS and non-CICS environments to assist you in problem diagnosis. For details, see *LE/VSE Installation and Customization Guide*.

Specifying Abnormal Termination Exit under Non-CICS: For non-CICS applications, the sample batch abnormal termination exit is CEEBXITA found in the PRD2.SCEEBASE sublibrary. This exit provides system abend dumps.

Specifying Abnormal Termination Exit under CICS: For CICS applications, the sample abnormal termination exit is CEECXITA also found in the PRD2.SCEEBASE sublibrary. This exit provides transaction dumps.

3.2 Chapter 5. Moving from the DOS/VS COBOL Run Time to LE/VSE

This chapter provides detailed information about running DOS/VS COBOL programs under LE/VSE. It includes information on:

- Determining which DOS/VS COBOL programs require link-edit with LE/VSE
- Determining which DOS/VS COBOL programs require compile with COBOL/VSE
- Comparing run-time options and specification mechanisms
- Converting programs using STXIT AB/STXIT PC for Condition Handling
- Running in the reusable run-time environment
- Managing dump services
- Using SORT/MERGE in DOS/VS COBOL programs
- Communicating with other languages (ILC)

Each section in this chapter indicates how that particular section is

applicable to applications compiled with batch and for applications running under CICS.

Subtopics:

3.2.1 Determine which DOS/VS COBOL Programs Require Link-Edit with LE/VSE

BATCH: Yes

CICS: No

DOS/VS COBOL programs that will be dynamically called by VS COBOL II or COBOL/VSE programs, must be relinked with LE/VSE.

3.2.2 Determine which DOS/VS COBOL Programs Require Compile with COBOL/VSE

BATCH: Yes

CICS: No

In order to determine which DOS/VS COBOL programs you need to compile with COBOL/VSE, you need to know if the program uses ILC with other high-level languages.

Additionally, applications that need to use LE/VSE or COBOL/VSE features (such as LE/VSE callable services, intrinsic functions, or calls to assembler) must be compiled with COBOL/VSE.

ILC Used: Interlanguage communication is defined as programs that call, or are called by, other high-level languages (PL/I). Assembler is not considered a high-level language; thus, calls to and from assembler programs are not considered ILC.

The following table shows the action required for applications with existing ILC:

Figure 17. Action Required for Existing Applications using ILC

High-Level Language	Comments
PL/I	DOS/VS COBOL programs calling or being called by PL/I must be compiled with COBOL/VSE.

3.2.3 Compare Run-Time Options and Specification Mechanisms

BATCH: Yes

CICS: No. Only the DOS/VS COBOL portion of the LE/VSE library is used. Therefore you can only use the options available to you under DOS/VS COBOL.

This section lists the mechanisms available to specify LE/VSE run-time options and gives a brief description of each method. It also shows what support, if any, LE/VSE provides for DOS/VS COBOL run-time options.

Subtopics:

3.2.3.1 Specify LE/VSE Run-Time Options

LE/VSE provides various methods for specifying run-time options. Three mechanisms are available to specify options:

- Run-time option CSECTs
  - CEEDOPT for installation-wide defaults
  - CEEUOPT for application-specific defaults
- Invocation procedures
- Assembler user exit

To determine which of the LE/VSE methods you can use for existing applications, you need to know if the main program has been link-edited with LE/VSE. Figure 18 lists the specification methods available to specify and replace run-time options under LE/VSE.

Figure 18. Methods Available for Specifying Run-Time Options for DOS/VS COBOL Programs

Main Program	CEEDOPT	Invocation	Default

			Assembler User Exit
DOS/VS COBOL not relinked		X(2)	
DOS/VS COBOL relinked (no COBOL/VSE)		X(2)	
DOS/VS COBOL relinked with COBOL/VSE(1)	X	X(3)	X

Note:

(1) Normally, link-editing DOS/VS COBOL programs with LE/VSE has no effect. However, if a COBOL/VSE program is introduced into an application with all DOS/VS COBOL programs, after link-editing the programs with LE/VSE, the programs have access to CEEDOPT and the default assembler user exit.

(2) Via JCL OPTIONS/UPSI statements

(3) Via JCL PARM option

For more information on implications when adding COBOL/VSE programs to existing applications, see Chapter 12, "Exploiting LE/VSE by Adding COBOL/VSE Programs to Existing Applications" in topic 5.1.

Subtopics:

3.2.3.1.1 CEEDOPT Installation-Wide Defaults

LE/VSE provides an assembler file, CEEDOPT.A, that contains default values for all LE/VSE run-time options. At installation time, you can edit this file and select defaults that will apply to all applications running in the common environment. After editing, the file is assembled and link-edited with LE/VSE.

You can select a "nonoverridable" (fixed) attribute for options within this module. This allows your installation to enforce options that might be critical to your overall operating environment.

3.2.3.1.2 CEEUOPT Application-Specific Defaults

CEEUOPT is not available to DOS/VS COBOL programs, regardless of whether you have link-edited them with LE/VSE or not.

3.2.3.1.3 Invocation Procedures

Run-time options can also be specified at program invocation using the appropriate operating system mechanism.

For a DOS/VS COBOL application with no VS COBOL II or COBOL/VSE programs linked with it, the options AIXBLD / NOAIXBLD and DEBUG / NODEBUG specified by the **// OPTION SYSPARM** statement, and the UPSI compiler option is specified by the **// UPSI** JCL statement.

If a VS COBOL II or COBOL/VSE program is linked with the DOS/VS COBOL program, the options are specified via the PARM option of the JCL EXEC statement, eg. **// EXEC pgmname,PARM='/AIXBLD'**

For specific details, see the *LE/VSE Programming Guide*.

3.2.3.1.4 Order of Precedence

It is possible to use all of the above mechanisms for specifying run-time options for a single application. LE/VSE enforces the following rules of precedence (ordered from lowest to highest):

1. Installation-wide defaults defined at installation time
2. Application-specific options (link-edited with the application)
3. Options specified on invocation
4. Options specified through the Assembler User Exit
5. Installation-wide defaults specified as nonoverridable at installation time

---

3.2.3.2 Compare DOS/VS COBOL and LE/VSE Run-Time Options

The following table describes the support LE/VSE provides for DOS/VS COBOL run-time options.

Figure 19. Comparison of DOS/VS COBOL and LE/VSE Run-Time Options

DOS/VS COBOL Option	Comment
AIXBLD	AIXBLD is supported under LE/VSE as in DOS/VS COBOL.
DEBUG	DEBUG is supported under LE/VSE as in DOS/VS COBOL.
UPSI	UPSI is supported under LE/VSE as in DOS/VS COBOL.

3.2.4 Convert Programs Using STXIT AB/STXIT PC for Condition Handling

BATCH: Yes

CICS: No

DOS/VS COBOL applications can contain user established condition handling routines written in assembler. Normally, you would code an assembler routine (this is your STXIT AB routine) to issue a STXIT AB and register a STXIT AB exit. When an abend occurs, the STXIT AB exit receives control.

Note: Although this section references STXIT AB, this information also applies to assembler routines that set STXIT PCs and register STXIT PC exits.

Existing DOS/VS COBOL-only applications will work without change. However, if a COBOL/VSE program is present in a run unit with user-established condition-handling routines, it will **not** work under LE/VSE, since the LE/VSE STXIT AB routine receives control in the event of errors, program interrupts, and abends. You must convert all programs in an application that have user-established condition handling routines using STXIT ABs to COBOL/VSE.

Error Handling Routines Written for DOS/VS COBOL Programs: The entire application must be converted to COBOL/VSE.

Establishing User Condition Handling Routines with LE/VSE: With COBOL/VSE and LE/VSE, you can use the PROCEDURE-POINTER data item (an IBM extension) in conjunction with the SET statement to establish your own condition handling routine using LE/VSE-provided condition handling callable services. The user-established condition handling routines receive control before the LE/VSE default condition handling. You can write condition handling routines in either COBOL/VSE or LE/VSE-enabled assembler.

Advantages of LE/VSE User Condition Handling Routines: Managing the point where you resume execution after handling an error is much simpler with LE/VSE than with DOS/VS COBOL. Under both DOS/VS COBOL and LE/VSE, recursive calls are not allowed. Generally, this means that after intercepting an error, the condition handler must resume at the Next Sequential Instruction (NSI) following the instruction that incurred the error.

When running COBOL/VSE or LE/VSE-enabled assembler programs under LE/VSE, LE/VSE automatically deactivates your programs when you change the resume point.

Conversion Scenario: In order to convert your existing error handling routines to the LE/VSE-style, you need to:

1. Separate your STXIT AB exit into a subroutine and use COBOL to get control of your STXIT AB exit.

2. Replace the call to the STXIT AB routine with a call to the LE/VSE callable service CEEHDLR.

3. Make your STXIT AB exit into a separate assembler routine.

4. Change the logic of your condition handler. You must use either LE/VSE services to indicate whether you want to RESUME, PERCOLATE (let another handler take control), or PROMOTE (change the condition to another condition). You can also use LE/VSE services to find the name of the program that incurred the condition, or retrieve the error message associated with the condition.

3.2.5 Run in the Reusable Run-Time Environment

BATCH: No

CICS: No

COBOL/VSE and LE/VSE continue to provide compatibility support for the reusable run-time environment for applications provided by ILBDSET0.

When using the reusable environment with existing applications with LE/VSE, the following restrictions apply:

- The main program must be a COBOL program
- The application running in the reusable environment can contain only COBOL and assembler language programs
- If an assembler driver is statically linked with ILBDSET0 to provide the reusable environment, it must be link-edited with the LE/VSE run-time library

Using STOP RUN will end the reusable environment regardless of the method used to initialize it. If ILBDSET0 is called by an assembler program, control is returned to the caller of the assembler program.

Using ILBDSET0: ILBDSET0 is still supported so that existing applications that use non-COBOL programs as main programs will continue to run and provide the same results.

3.2.6 Manage Dump Services

BATCH: No

CICS: Yes

Dump services are managed by LE/VSE. This section describes the support for symbolic dumps and system dumps. It also includes information on the LE/VSE formatted dump.

Subtopics:

3.2.6.1 DOS/VS COBOL Symbolic Dumps

Dump output generated by DOS/VS COBOL, using SYMDMP, remains the same and will be directed to SYSLST.

3.2.6.2 System Storage Dumps

To obtain a system storage dump, you need to use the abnormal termination exit. For details, see "Specify Abnormal Termination Exit for System Dump" in topic 3.1.4.

With DOS/VS COBOL, dump output was directed to SYSLST.
With LE/VSE system storage dump is directed to CEEDUMP or SYSLST.

3.2.6.3 LE/VSE Formatted Dumps

DOS/VS COBOL did not produce a formatted dump. With LE/VSE, when a condition of 2 or greater is raised and left unhandled, you can produce a formatted dump by specifying the TERMTHDACT(DUMP) run-time option.

The LE/VSE formatted dump contains abend information, SNAP dump information, and formatted dump information. It is intended to help you in debugging and problem determination. For examples of LE/VSE dump output, see *LE/VSE Debugging Guide and Run-Time Messages*.

With LE/VSE, you can indicate the destination for dump output by providing the CEEDUMP DLBL in your JCL. If the CEEDUMP file is not defined, LE/VSE will write the dump output to SYSLST.

For the attributes of the CEEDUMP file, see the *LE/VSE Programming Guide*.

Note: On CICS, all run-time output from LE/VSE is directed to a CICS transient data queue named CESE.

3.2.7 Use SORT/MERGE in DOS/VS COBOL Programs

BATCH: No

CICS: No

Implementation of LE/VSE caused changes in the run-time routines that manage DOS/VS COBOL SORT/MERGE. If a DOS/VS COBOL program initialized the SORT/MERGE, the following changes apply:

- LE/VSE will not produce an LE/VSE dump if a program check or abnormal termination occurs while in a SORT/MERGE user exit
- DOS/VS COBOL debug data will not be produced if a program check or abnormal termination occurs while in a SORT/MERGE user exit
- LE/VSE will not perform environment clean-up if a program check or abnormal termination occurs while in a SORT/MERGE user exit
- While in an input or output procedure, if an assembler program is called, the assembler program cannot call a COBOL program

If a SORT or MERGE statement is executed from a DOS/VS COBOL program, and the SORT/MERGE product issues a STXIT AB or STXIT PC, and does not reset the STXIT AB and/or STXIT PC upon return, the TRAP(ON) option will be disabled.

3.2.8 Communicate with Other Languages

BATCH: Yes

CICS: No. Current ILC on CICS is unaffected, except that ILC within the same run unit is not allowed.

Existing applications that use interlanguage communication (ILC) might not be able to run with LE/VSE. There are several determining factors, including which languages are involved. The following sections describe the implications for existing applications with ILC.

Subtopics:

3.2.8.1 COBOL and FORTRAN

This is not supported.

3.2.8.2 COBOL and PL/I

LE/VSE supports existing ILC between the following combinations of COBOL and PL/I, provided you follow the link-edit requirements below:

- VS COBOL II
- COBOL/VSE
- PL/I for VSE

ILC between DOS/VS COBOL and PL/I is not supported. Any DOS/VS COBOL programs containing ILC with PL/I must be compiled with COBOL/VSE.

For additional information on interlanguage communication between COBOL and PL/I, see *LE/VSE Programming Guide* and the *PL/I VSE Migration Guide*.

3.3 Chapter 6. Moving from the VS COBOL II Run Time to LE/VSE

This chapter describes how you can have compatibility and equivalent run-time results for your existing VS COBOL II and DOS/VS COBOL programs that have been running using the VS COBOL II run time. It includes information on:

- Determining which VS COBOL II programs require link-edit with LE/VSE
- Determining which VS COBOL II programs require compile with COBOL/VSE
- Comparing run-time options and specification methods
- Converting programs using STXIT AB/STXIT PC exits for condition handling
- Running in a reusable environment
- Initializing the run-time environment
- Determining storage tuning changes
- Locating return codes
- Managing messages and dump services
- Using SORT/MERGE with DOS/VS COBOL programs
- Communicating with other languages

In the following sections, DOS/VS COBOL programs are considered as NORES programs because they have the run time link-edited into the phase.

Subtopics:

3.3.1 Determine which VS COBOL II Programs Require Link-Edit with

LE/VSE

RES: Yes

NORES: Yes

CICS: No

Applications Compiled RES: If you call ILBDSET0 to establish a reusable environment (that is, you establish an assembler program as the main program by link-editing it with ILBDSET0), you must link-edit the assembler program with LE/VSE.

VS COBOL II programs calling DL/I and including DLZBPJRA, must be relinked as DLZBPJRA calls ILBDSET0.

If the main program is VS COBOL II, DLZPBJRA does not need to be included when running with either the VS COBOL II or LE/VSE run time. As a preparation for migration, these phases can be relinked to not include DLZPBJRA prior to migrating to LE/VSE.

If your programs compiled RES do not call ILBDSET0, they do not require link-edit with LE/VSE.

Applications Compiled NORES: If you have used the VS COBOL II MIXRES run-time option to mix both RES and NORES programs in an application, you must link-edit the following programs with LE/VSE:

° VS COBOL II NORES programs in a reusable environment

When relinking VS COBOL II NORES programs with LE/VSE, an INCLUDE statement for IGZENRI should be included before the link-edit step. This will ensure the correct versions of the library routines are included. If this is not done, the phase will be larger and messages as follows indicating duplicate sections will be issued during the link-edit step.

```
2139I DUPLICATE SECTION DEFINITION:  IGZCDSP . ***** SECTION IGNORED
```

For additional details on applications mixing RES programs and NORES programs, see Appendix D, "Mixed RES and NORES Applications" in topic APPENDIX1.4.

3.3.2 Determine which VS COBOL II Programs Require Compile with COBOL/VSE

RES: Yes

NORES: Yes

CICS: No

Existing VS COBOL II programs do not require compile with COBOL/VSE in order to run under LE/VSE. However, if you need to use LE/VSE or COBOL/VSE features (such as LE/VSE callable services, COBOL/VSE intrinsic functions, or calls to LE/VSE-enabled assembler), you must compile with COBOL/VSE.

3.3.3 Compare Run-Time Options and Specification Methods

RES: Yes

NORES: Yes

CICS: Yes

VS COBOL II run-time options can be affected when running existing applications with LE/VSE. The following sections describe the differences that can occur. For specific details relating to run-time options new with LE/VSE and existing run-time options supported under LE/VSE, see the *LE/VSE Programming Guide*.

Subtopics:

3.3.3.1 Specify LE/VSE Run-Time Options

LE/VSE provides various methods for specifying run-time options. Three mechanisms are available to specify options:

- Run-time option CSECTs
  - CEEDOPT for installation-wide defaults (non-CICS)
  - CEECOPT for installation-wide defaults (CICS)
  - CEEUOPT for application-specific defaults
- Invocation procedures
- Assembler user exit

In addition, the VS COBOL II application-specific run-time options module (IGZEOPT) is available to VS COBOL II applications compiled RES, which do not run under CICS.

To determine which of the LE/VSE methods you can use for existing applications, you need to know whether the main program is compiled with RES or NORES, and if the main program has been link-edited with LE/VSE. Figure 20 lists the specification methods available to specify and replace run-time options under LE/VSE.

Figure 20. Methods Available for Specifying Run-Time Options

Main Program	CEEDOPT	CEEUOPT	Invocation	Default Assembler User Exit	IGZEOPT
Not link-edited with LE/VSE:					
DOS/VS COBOL			X		
VS COBOL II NORES			X		X
VS COBOL II RES	X	X	X	X	X
Link-edited with LE/VSE:					
DOS/VS COBOL (1)			X		
VS COBOL II NORES	X	X	X	X	
VS COBOL II RES	X	X	X	X	X

Note:

(1) Normally, link-editing DOS/VS COBOL programs with LE/VSE has no effect. However, if you introduce a COBOL/VSE or VS COBOL II program into an application with all DOS/VS COBOL programs, after link-editing the DOS/VS COBOL programs now exhibit RES behavior and have access to CEEDOPT and the default assembler user exit.

For more information on implications when adding COBOL/VSE programs to existing applications, see Chapter 12, "Exploiting LE/VSE by Adding COBOL/VSE Programs to Existing Applications" in topic 5.1.

Subtopics:

3.3.3.1.1 CEEDOPT Installation-Wide Defaults

LE/VSE provides an assembler file, CEEDOPT.A, that contains default values for all LE/VSE run-time options. At installation time, you can edit this file and select defaults that will apply to all applications running in the common environment. After editing, the file is assembled and link-edited with LE/VSE.

You can select a "nonoverridable" (fixed) attribute for options within this module. This allows your installation to enforce options that might

be critical to your overall operating environment.

3.3.3.1.2 CEEUOPT Application-Specific Defaults

LE/VSE provides an assembler file, CEEUOPT.A, that you can edit to select run-time options that will apply to a specific application or program.

After editing, the file is assembled and link-edited with the specific application or program to which it applies. The options you set with this module take precedence over any overridable installation-wide options set with CEEDOPT.

3.3.3.1.3 Invocation Procedures

Run-time options can also be specified using the PARM option of the JCL EXEC statement.

For DOS/VS COBOL-only programs, the JCL OPTION statement and JCL UPSI statement are used.

Note: Under CICS, you cannot use the PARM parameters to specify run-time options. Instead, use CEECOPT for installation-wide default run-time option settings. And, as with non-CICS applications, use CEEUOPT for application-specific run-time option settings.

For specific details, see the *LE/VSE Programming Guide*.

3.3.3.1.4 Order of Precedence

It is possible to use all of the above mechanisms for specifying run-time options for a single application. LE/VSE enforces the following rules of precedence (ordered from lowest to highest):

1. Installation-wide defaults defined at installation time
2. Application-specific options (link-edited with the application)
3. Options specified on invocation
4. Options specified through the Assembler User Exit
5. Installation-wide defaults specified as nonoverridable at installation time

---

3.3.3.2 Specifying VS COBOL II Run-Time Options

The following information is included only as a compatibility aid to assist you in running existing applications without having to compile with COBOL/VSE or link-edit with LE/VSE. As programs in the application are modified, it is recommended that the appropriate LE/VSE options module be included in the application.

Applications Compiled RES (non-CICS): The application-specific VS COBOL II run-time options module, IGZEOPT, is available to RES applications, running under LE/VSE. IGZEOPT is not affected when link-editing the application with LE/VSE, therefore it is possible for both IGZEOPT and CEEUOPT to exist in the same application. Figure 21 shows the relationship between IGZEOPT and CEEUOPT:

Figure 21. Application-specific Options for Existing Applications

IGZEOPT Present	**CEEUOPT Present**	**Comments**
No	No	The run-time options are not affected.
Yes	No	The run-time options specified in IGZEOPT are mapped to the appropriate LE/VSE run-time option.
No	Yes	The run-time options specified in CEEUOPT are in effect.
Yes	Yes	The run-time options specified in CEEUOPT are in effect. IGZEOPT is ignored. No error message is issued.

You should replace IGZEOPT with CEEUOPT when you link-edit these applications with LE/VSE.

Applications Compiled NORES (non-CICS): The default VS COBOL II run-time options module, IGZEOPD, is available to NORES applications that have been link-edited with the VS COBOL II run-time library. Once these applications are link-edited with LE/VSE, IGZEOPD is no longer available.

3.3.3.3 Compare VS COBOL II and LE/VSE Run-Time Options

The following table describes how VS COBOL II run-time options are either supported or changed when running with LE/VSE. Synonym indicates that the VS COBOL II option is mapped to an LE/VSE option. The differences in behavior (if any) are described in the comments column.

Figure 22. Comparison of VS COBOL II and LE/VSE Run-Time Options

VS COBOL II Option	**LE/VSE Synonym**	**Comment**
AIXBLD	AIXBLD	With LE/VSE, the Access Method Services (AMS) messages are directed to the file specified on the LE/VSE run-time option MSGFILE (default is SYSLST). AIXBLD is not applicable under CICS.
DEBUG	DEBUG	This option is specified and supported in the same manner under LE/VSE as in VS COBOL II.

LANGUAGE	NATLANG	NATLANG replaces LANGUAGE, which is a VS COBOL II installation option. However, LE/VSE provides the ability to select a national language at run time, as well as at installation time, using the NATLANG option.
LIBKEEP		LIBKEEP is not supported. For similar performance function use the LE/VSE pre-initialization services. For more information, see "Initialize the Run-Time Environment" in topic 3.3.6. The LIBKEEP option is not applicable under CICS.
MIXRES		MIXRES is not supported under LE/VSE. For more information, see Appendix D, "Mixed RES and NORES Applications" in topic APPENDIX1.4. The MIXRES option is not applicable under CICS.
RTEREUS	RTEREUS	RTEREUS is supported in the same manner under LE/VSE as in VS COBOL II. The RTEREUS option is not applicable under CICS.
SPOUT	RPTOPTS(ON) RPTSTG(ON)	SPOUT is mapped to the LE/VSE options RPTOPTS and RPTSTG. Storage reports are directed to the filename specified in MSGFILE (default SYSLST). In addition, the report formats are different. For more information, see "Determine Storage Tuning Changes" in topic 3.3.7.
SSRANGE	CHECK(ON)	SSRANGE is mapped directly to CHECK.
STAE	TRAP(ON)	STAE is mapped directly to TRAP. For non-CICS applications, TRAP causes both STXIT AB and STXIT PC to be issued. Under VS COBOL II, STAE only causes STXIT AB to be issued.
UPSI	UPSI	This option is specified and supported in the same manner in LE/VSE as in VS COBOL II.
WSCLEAR	STORAGE	WSCLEAR is not supported. For similar function, use STORAGE(00,NONE,NONE,8K) for non-CICS applications and STORAGE(00,NONE,NONE,0K) for CICS applications. For more information, see "Determine Storage Tuning Changes" in topic 3.3.7.

3.3.4 Convert Programs Using STXIT AB/STXIT PC Exits for Condition Handling

RES: Yes

NORES: Yes

CICS: No

VS COBOL II and DOS/VS COBOL applications can contain user established condition handling routines written in assembler. Normally, you would code an assembler routine (this is your STXIT AB routine) to issue a STXIT AB which registers a STXIT AB exit. When an abend occurs, the STXIT AB exit receives control.

Note: Although this section references STXIT AB, this information also applies to assembler routines that issue STXIT PCs and register STXIT PC exits.

When running programs under LE/VSE, the LE/VSE condition manager receives control of errors, program interrupts, and abends. Existing user-established condition handling routines do not work under LE/VSE. You either need to convert the entire application to COBOL/VSE, or rewrite parts of your existing condition handling routines to the LE/VSE-style of condition handling, depending on whether the error handling routines are written for VS COBOL II programs.

Error Handling Routines Written for VS COBOL II Programs: For a VS COBOL II application, total source conversion is not necessary. You only need to convert programs that either establish or perform condition handling. You will need to:

1. Separate your STXIT AB exit into a subroutine and use COBOL to get control of your STXIT AB exit

2. Replace the call to the STXIT AB routine with a call to CEEHDLR (an LE/VSE callable service)

3. Make your STXIT AB exit into a separate LE/VSE-enabled routine (COBOL/VSE or LE/VSE-enabled assembler)

4. Change the logic of your condition handler. You must use LE/VSE services to indicate whether you want to RESUME, PERCOLATE (let another handler take control), or PROMOTE (change the condition to another condition). You can also use LE/VSE services to find the name of the program that incurred the condition, or retrieve the error message associated with the condition.

Establish User-Written Condition Handling Routines with COBOL/VSE: With COBOL/VSE, you can use the PROCEDURE-POINTER data item (an IBM extension) in conjunction with the SET statement to establish your own condition handling routine using LE/VSE-provided condition handling callable services. The user-established condition handling routines receive control before the LE/VSE default condition handling. You can write condition handling routines in either COBOL/VSE or LE/VSE-enabled assembler.

Advantages of LE/VSE User Condition Handling Routines: Managing the point

where you resume execution after handling an error is much simpler with LE/VSE, than with VS COBOL II. Under VS COBOL II and LE/VSE, recursive calls are not allowed. Generally, this means that after intercepting an error, the condition handler must resume at the Next Sequential Instruction (NSI) following the instruction that incurred the error.

With VS COBOL II, in order to resume in a program other than the one that incurred the error, you are required to delete the programs that are active at the time of the error and that might be re-entered after the STXIT AB exit received control.

When running COBOL/VSE or LE/VSE-enabled assembler programs under LE/VSE, LE/VSE automatically deactivates your programs when you change the resume point.

For VS COBOL II programs, the move resume cursor function (LE/VSE callable service CEEMRCR) will cause stack frame collapse, where programs that were active are rendered inactive, and can then be re-entered. This means that you can do any of the following:

- Resume at the Next Sequential Instruction (NSI) of the failing program
- Move the resume cursor to the instruction following the CALL statement to the failing program
- Move the resume cursor to the NSI of the CALLer of the CALLer of the failing program

---

3.3.5 Running in a Reusable Environment

RES: Yes

NORES: No

CICS: No

LE/VSE continues to provide compatibility support for the existing methods for managing the reusable run-time environment. Three ways in which you can establish a reusable environment are:

- IGZERRE programs
- ILBDSET0 programs
- RTEREUS run-time option

Restrictions when Running under LE/VSE: The following restrictions apply to existing applications using the reusable environment and running under LE/VSE:

- The application must be a single enclave (phase) application
- The main program must be a COBOL program

- The application can contain only COBOL and assembler language programs

- If an assembler driver is statically linked with IGZERRE or ILBDSET0 to provide the reusable environment, it must be link-edited with the LE/VSE run-time library

End the Reusable Environment: Using STOP RUN will end the reusable environment regardless of the method used to initialize it. If IGZERRE or ILBDSET0 is called by an assembler program, control is returned to the caller of the assembler program. If you use the LE/VSE RTEREUS run-time option, control is returned to the caller of the caller of the first COBOL program.

Subtopics:

3.3.5.1 Use IGZERRE

You can continue to use IGZERRE to explicitly drive the initialization and termination functions of COBOL. However, you need to be aware of some changes:

On return from IGZERRE, register 15 contains a return code as shown below.

Figure 23. Return Code Changes for IGZERRE

Return Code	Comments
0	No change.
4	Is issued if LE/VSE is already initialized (previously issued if VS COBOL II was already initialized).
8	No change.
12	Is no longer issued (applied to initialization of a NORES environment only).
16	No change.
20	Is no longer issued (applied to use of COBTEST).

For details on using IGZERRE including register contents, return codes, and rules for the RES environment, see the *VS COBOL II Application Programming Guide for VSE*.

3.3.5.2 Use ILBDSET0

ILBDSET0 is supported so that existing applications that use non-COBOL programs as main programs will continue to run and provide the same results.

If you use ILBDSET0 to initialize the COBOL reusable environment, LE/VSE will automatically set ALL31(OFF) and STACK(,,BELOW), as long as the application-specific routine CEEUOPT is **not** linked with the phase.

If you link CEEUOPT with the phase, then you must ensure that it specifies ALL31(OFF) and STACK(,,BELOW).

In the event that the installation-wide default routine CEEDOPT specifies ALL31(ON) and/or STACK(,,ANYWHERE) as nonoverridable, then LE/VSE diagnoses this conflict with message CEE3615I. Running the application with these conflicts will produce unpredictable results.

3.3.5.3 Use RTEREUS

The RTEREUS run-time option for initializing the COBOL reusable environment is supported by LE/VSE. For RES environments, the behavior of RTEREUS is the same as it was in VS COBOL II.

If the operating system is the invoker of the first COBOL program, RTEREUS will be ignored. If the LE/VSE environment is already initialized, the reusable environment is not supported.

You can specify RTEREUS using either the installation-wide default routine CEEDOPT, or the application-specific options routine CEEUOPT.

3.3.6 Initialize the Run-Time Environment

RES: Yes

NORES: No

CICS: No

The VS COBOL II LIBKEEP run-time option was used to enhance run-time performance by maintaining the partition level of the run-time environment between calls to COBOL main programs. The LE/VSE run-time environment does not support the LIBKEEP function.

You can obtain similar performance for the main program environment by using the pre-initialization services provided by LE/VSE. These services enable you to create and initialize an LE/VSE environment, run COBOL main programs multiple times within the environment, and control environment termination.

Subtopics:

3.3.6.1 Existing Applications Using LIBKEEP

LE/VSE pre-initialization services for the main program environment allow you to initialize the run-time environment for a main program by using the CEEPIPI(INIT_MAIN,...) service, to call programs as main by using the CEEPIPI(CALL...) service, and to terminate the pre-initialized environment using the CEEPIPI(TERM...) service.

To use the LE/VSE pre-initialization main program services in place of LIBKEEP, your application **cannot** contain any DOS/VS COBOL programs. The presence of any DOS/VS COBOL programs will produce unpredictable results. Also, a VS COBOL II program cannot be the target of CEEPIPI(CALL...). However, it can be the target of any COBOL CALL statements.

Subtopics:

3.3.6.1.1 Differences

Previously, when running an application with the run-time option LIBKEEP, ACCEPT SYSIPT, DISPLAY SYSLST, and DISPLAY SYSPCH were closed after the completion of each invocation of a main program. Also, the run-time messages and dumps could have been received at the end of each execution.

When running with the LE/VSE pre-initialization facility for the main program environment, those files plus MSGFILE and dump files will not be closed until CEEPIPI(TERM) is issued to terminate the pre-initialized environment. Main programs called repeatedly under the LIBKEEP environment are considered as separate applications, while the main programs called repeatedly under the LE/VSE pre-initialization environment are considered to be within a single application and the application is not completed until CEEPIPI(TERM) is issued.

For more information on LE/VSE pre-initialized services, including information on how to initialize the run-time environment for subprograms, see the *LE/VSE Programming Guide*.

3.3.7 Determine Storage Tuning Changes

RES: Yes

NORES: No

CICS: Yes

Space management and tuning is now controlled by LE/VSE services. Existing applications that use space tuning and management facilities such as the IGZTUNE macro, and the SPOUT and WSCLEAR run-time options, will use the new LE/VSE facilities as described in the following sections.

For details and specifics of space management and tuning with LE/VSE, see the *LE/VSE Programming Guide*.

Subtopics:

3.3.7.1 When IGZTUNE Is Not Supported

If the CSECT produced by assembling the space tuning CSECT IGZETUN is detected in a phase, a warning level message is issued and the CSECT is not used. Other LE/VSE facilities provide storage management capabilities at run time instead of link-edit time. This is accomplished by using the following five storage management run-time options:

HEAP Manages heap storage for user data such as working storage, external data, and external file information.

ANYHEAP Manages heap storage, which can be located anywhere and used for data control blocks, for use by LE/VSE and COBOL library routines.

BELOWHEAP Manages heap storage, which is located below the 16-megabyte line and used for Data Control Blocks and I/O buffers, for use by LE/VSE and COBOL library routines,

STACK Manages stack storage, which can be used for Dynamic Storage Area (DSAs), for use by LE/VSE and COBOL library routines,

LIBSTACK Manages stack storage, which is located below the 16-megabyte line and used for DSAs, for use by LE/VSE and COBOL library routines,

The RPTSTG run-time option provides the optimum values to use when specifying these options. (You will need to first use the RPTSTG option to generate a report of storage used in your program or run unit. You can then use this report to determine the values you need to specify in the five LE/VSE storage options in order to achieve the space tuning purpose.)

Do not use the values from IGZTUNE when specifying the LE/VSE tuning options. LE/VSE performs storage tuning and management in a manner different from VS COBOL II. For recommended LE/VSE storage option settings, see "Set Recommended Default LE/VSE Run-Time Options" in topic 3.1.1.

3.3.7.2 SPOUT Output

Existing applications that are specified with the SPOUT run-time option will continue to run. The SPOUT option is mapped to the LE/VSE run-time options RPTOPTS and RPTSTG.

The output is directed to the filename specified in the MSGFILE run-time option. The default is SYSLST. You can use the information generated by the RPTOPTS and RPTSTG options to determine the values to use when tuning storage with the LE/VSE storage options.

3.3.8 Locate Return Codes

RES: Yes

NORES: No

CICS: Only information on Return from a COBOL Subprogram.

LE/VSE uses different return codes and return code modifier schemes than VS COBOL II. The relationship between the LE/VSE format and the COBOL RETURN-CODE special register is described below.

Return from a COBOL Subprogram: When returning from a COBOL subprogram, the relationship remains the same as in VS COBOL II. The COBOL RETURN-CODE is placed in R15 at termination.

Return from an Enclave: An enclave (phase) return code (which includes the COBOL RETURN-CODE) and an LE/VSE return code modifier will be returned to the invoker of an enclave when the invoked enclave terminates. The invokers, including those of the first enclave, will receive the enclave return code in R15 and the LE/VSE return code modifier in R0.

For more information on return codes, see the *LE/VSE Programming Guide*.

3.3.9 Manage Messages and Dump Services

RES: Yes

NORES: No

CICS: Yes

Messages and dump services are managed differently for existing applications running with LE/VSE than when running with the VS COBOL II library.

Subtopics:

3.3.9.1 Run Time Message Management

Two factors determine how run-time messages for existing applications are managed when running with LE/VSE:

- If the messages are issued by the VS COBOL II compatibility library routines (Format CnnnI, followed by the message text)
- Status of the LE/VSE run-time initialization process for messages issued by COBOL-specific library routines, including VS COBOL II compatibility routines (prefixed IGZ)

Subtopics:

3.3.9.1.1 Messages with Format CnnnI

There are no differences for DOS/VS COBOL messages. The message prefix, number, severity, and content remain unchanged. Also, the destination remains the same, even when the program is link-edited with LE/VSE. The messages appear synchronously and are written to the console (SYSLOG). This is true regardless of the state of the run-time initialization process.

3.3.9.1.2 Messages Prefixed with IGZ

VS COBOL II messages are managed depending on the status of the LE/VSE run-time initialization process.

- If the run-time initialization process is not complete, messages are routed as directed by OS write-to-programmer
- If the run-time initialization process is complete, then messages are directed to the file *filename* specified on the LE/VSE MSGFILE run-time option, which is defaulted to SYSLST

If MSGFILE filename is not defined, a severe condition will be be raised and a message will be written to SYSLOG. For details, see *LE/VSE Programming Guide*.

- On CICS, all run-time output from LE/VSE is directed to a CICS transient data queue named CESE. Each record written to the file has a header that includes the terminal-ID, the transaction-ID, and date and time. Run-time output is no longer written to a temporary storage queue

All COBOL-specific messages will be prefixed by IGZ, followed by a 4-digit message number, and the level of severity. For example, VS COBOL II message number IGZ020I is message number IGZ0020W under LE/VSE. The VS COBOL II informational message now becomes severity 1 warning message. The VS COBOL II severe message, which then resulted in an abend, now becomes either severity 3 severe condition or severity 4 critical condition as explained below:

Severity 1 (W)
is a warning message. The LE/VSE default condition handling action will format and issue the message and then continue program execution.

Severity 3 (S)
is a severe error message. The LE/VSE default condition handling action will percolate the condition, issue a message, and terminate the enclave.

Severity 4 (C)
is a critical error message. LE/VSE uses this message to diagnose errors caused by COBOL-specific library routines. The LE/VSE default condition handling action will percolate the condition, issue a message, and terminate the enclave.

For detailed MSGFILE attributes, see the *LE/VSE Programming Guide*.

National Language Support: LE/VSE provides run-time options and callable services that allow you to dynamically select the national language for error messages, and country settings for the date/time in error messages, dump output, and storage reports. COBOL/VSE and LE/VSE both support the following languages:

- Mixed-case American English
- Uppercase American English
- Japanese

Note: COBOL definitions for the currency symbol, decimal separator, and thousands separator are not affected by the National Language Support of LE/VSE.

3.3.9.2 Dump Services

Dump services are managed by LE/VSE. This section describes the changes for symbolic dumps, formatted dumps, and system dumps.

CICS Considerations: All run-time output from LE/VSE is directed to a CICS transient data queue named CESE. Each record written to the file has a header that includes the terminal ID, the transaction ID, and the date and time.

Subtopics:

3.3.9.2.1 DOS/VS COBOL Symbolic Dumps

Dump output generated by DOS/VS COBOL remains the same and is directed to SYSLST.

3.3.9.2.2 LE/VSE Formatted Dumps

Dump formats and destinations for dump output are different under LE/VSE than with VS COBOL II.

The LE/VSE formatted dump contains abend information, SNAP dump information, and formatted dump information intended to help you in debugging and problem determination. For examples of LE/VSE dump output, see *LE/VSE Debugging Guide and Run-Time Messages*.

With LE/VSE you can indicate the destination for dump output by providing the CEEDUMP DLBL in your JCL. For the attributes of the CEEDUMP file, see the *LE/VSE Programming Guide*.

If the CEEDUMP file is not defined, LE/VSE will direct output to SYSLST.

Under LE/VSE, specify the TERMTHDACT(DUMP) run-time option in order to get an LE/VSE formatted dump when a condition greater than or equal to 2 is raised and is left unhandled.

3.3.9.2.3 System Storage Dumps

To obtain a system storage dump, you need to use the abnormal termination exit. For details, see "Specify Abnormal Termination Exit for System Dump" in topic 3.1.4. With VS COBOL II, dump output is directed to SYSLST.

3.3.10 Use SORT/MERGE with DOS/VS COBOL Programs

RES: Yes

NORES: No

CICS: No

Implementation of LE/VSE caused changes in the run-time routines that manage VS COBOL II SORT/MERGE. If a VS COBOL II program initialized the SORT/MERGE, the following changes apply:

- LE/VSE will not produce a run-time dump if a program check or abnormal termination occurs while in a SORT/MERGE user exit
- VS COBOL II debug data will not be produced if a program check or abnormal termination occurs while in a SORT/MERGE user exit
- LE/VSE will not perform environment clean-up if a program check or abnormal termination occurs while in a SORT/MERGE user exit
- While in an input or output procedure, if an assembler program is called, the assembler program cannot call a COBOL program

---

3.3.11 Communicate with Other Languages

RES: Yes

NORES: Yes

CICS: No. Current ILC on CICS is unaffected, except that ILC within the same run unit is not allowed.

Existing applications that use interlanguage communication (ILC) might not be able to run with LE/VSE. There are several determining factors, including which languages are involved. The following sections describe the implications for existing applications with ILC.

Subtopics:

3.3.11.1 COBOL and FORTRAN

This is not supported.

3.3.11.2 COBOL and PL/I

LE/VSE supports existing ILC between the following combinations of COBOL and PL/I, provided you follow the link-edit requirements below:

- VS COBOL II Release 3.2 and later
- COBOL/VSE
- PL/I for VSE

Existing PL/I routines must be recompiled with PL/I VSE.

VS COBOL II applications need to be relinked with LE/VSE,

- When the COBOL application contains NORES programs.
- When the COBOL application requires program-specific run-time options or space tuning.

Note: ILC between DOS/VS COBOL and PL/I is not supported. Any DOS/VS COBOL programs containing ILC with PL/I must be compiled with COBOL/VSE.

For additional information on interlanguage communication between COBOL and PL/I, see *LE/VSE Programming Guide* and the *PL/I VSE Migration Guide*.

3.3.11.3 COBOL and C/370*

ILC between existing C/370 programs (those compiled by IBM C/370 Compiler Version 2 5688-187), and COBOL programs cannot run on LE/VSE.

In order to run existing applications that have ILC between COBOL programs and C programs, you must replace C and COBOL library objects.

4.0 Upgrading Your Source to COBOL/VSE

Subtopics:

4.1 Chapter 7. Modifying Your DOS/VS COBOL Source

Programs

This chapter describes the differences between the DOS/VS COBOL and the COBOL/VSE languages. The information in this chapter will help you evaluate, from a language standpoint, which COBOL applications are good candidates for upgrading to COBOL/VSE.

COBOL/VSE programs compiled with the NOCMPR2 compiler option provide COBOL 85 Standard-conforming programs, which is the strategic direction for the IBM COBOL language.

When upgrading your DOS/VS COBOL programs to COBOL/VSE, it is recommended that you convert them to NOCMPR2 programs in order to take advantage of the language enhancements provided by the COBOL 85 Standard, as well as to prepare for future Standard enhancements.

This chapter is not intended to be a syntax guide. You can find complete descriptions and coding rules for the relevant COBOL language elements in:

- *IBM VS COBOL for DOS/VSE*
- *COBOL/VSE Language Reference*

Notes:

1. There are special considerations for new, changed, or unsupported language elements when you are running under CICS. For details, see Chapter 11, "Migrating Subsystem Source Programs" in topic 4.5

2. In the following sections, any reference to COBOL 68 Standard is a reference to IBM Full American National Standard COBOL, Version 4 (Program 5736-CB2), or to LANGLVL(1) of DOS/VS COBOL(Program 5746-CB1)

3. The information provided in this chapter, and throughout this book, is intended for DOS/VS COBOL Release 3.1., with the latest service updates applied

Subtopics:

4.1.1 Compare DOS/VS COBOL to COBOL/VSE

DOS/VS COBOL supports the COBOL 68 Standard (LANGLVL(1)) and the COBOL 74 Standard (LANGLVL(2)). COBOL/VSE with the NOCMPR2 compiler option supports the COBOL 85 Standard. In addition to the language differences between the COBOL 74 Standard and COBOL/VSE with NOCMPR2, your DOS/VS COBOL LANGLVL(2) programs might contain undocumented DOS/VS COBOL extensions.

Subtopics:

4.1.1.1 Identify Language Elements that Require Change - Quick Reference

Figure 24 lists the language elements different in DOS/VS COBOL and COBOL/VSE. It also lists conversion aids, if any, available to automate the conversion or flag language elements that require manual conversion.

The language items listed in the table are described in detail throughout this chapter, and are classified and ordered according to the following categories:

- DOS/VS COBOL language elements not supported by COBOL/VSE
- DOS/VS COBOL language elements implemented differently in COBOL/VSE
- Undocumented DOS/VS COBOL extensions

Figure 24. Language Element Differences between DOS/VS COBOL and COBOL/VSE

Language Element	Conversion Aid	Page
ALPHABETIC class changes	CCCA	4.1.6
ALPHABET Clause changes - SPECIAL-NAMES paragraph	CCCA	4.1.6
APPLY WRITE-ONLY clause restrictions removed	None	4.1.6
Arithmetic statement changes	None	4.1.6
ASSIGN TO *integer system-name*	CCCA	4.1.4
ASSIGN ... FOR MULTIPLE REEL /UNIT	CCCA	4.1.4
ASSIGN clause changes -*assignment-name* forms	CCCA	4.1.6
B symbol in PICTURE clause--changes in evaluation	CCCA	4.1.6
BLANK WHEN ZERO clause and asterisk (*) replace	None	4.1.5
CALL identifier statement - B symbol in PICTURE clause	CCCA	4.1.6
CALL statement changes - procedure names and file names in USING option	CCCA	4.1.6
CANCEL statement - B symbol in PICTURE clause	CCCA	4.1.6
Combined Abbreviated Relation Condition changes	CCCA	4.1.6
COM-REG special register	none	4.1.4

COPY statement changes - 68 Standard implementation	None	4.1.6
CURRENCY-SIGN clause changes - in the SPECIAL-NAMES paragraph	CCCA	4.1.6
CURRENT-DATE special register	CCCA	4.1.4
DAM file handling	CCCA(1)	4.1.3.3
DIVIDE...ON SIZE ERROR - change in intermediate results	CCCA	4.1.6
EXAMINE statement	CCCA	4.1.4
EXHIBIT statement	CCCA	4.1.4
EXIT PROGRAM/GOBACK statement changes	CCCA	4.1.6
FILE STATUS clause changes	CCCA	4.1.6
FILE-LIMIT clause of the FILE-CONTROL paragraph	CCCA	4.1.4
FOR MULTIPLE REEL /UNIT	CCCA	4.1.4
GIVING option of USE AFTER STANDARD ERROR declarative	CCCA	4.1.4
IF...OTHERWISE statement changes	CCCA	4.1.6
Index names	CCCA	4.1.5
INITIALIZE...REPLACING--changes in evaluation	CCCA	4.1.6
ISAM file handling	CCCA	4.1.3.2
INSPECT statement - in PROGRAM COLLATING SEQUENCE clause	CCCA	4.1.6
JUSTIFIED clause changes	CCCA	4.1.6
LABEL RECORDS is data-name in non-sequential files	CCCA	4.1.4
MOVE statement - binary value vs display value	None	4.1.5
MOVE statements and comparisons--scaling changes	CCCA	4.1.6
MOVE CORRESPONDING statement	CCCA	4.1.5
MULTIPLY...ON SIZE ERROR - change in intermediate results	CCCA	4.1.6
Nonunique program-ID names	CCCA	4.1.5
NOTE statement	CCCA	4.1.4
NSTD-REELS special register	none	4.1.4
NUMERIC class tests for group items	none	4.1.4
OCCURS clause	CCCA	4.1.5
OCCURS DEPENDING ON - ASCENDING/DESCENDING KEY Option	None	4.1.6
OCCURS DEPENDING ON - Value for Receiving Items Changed	CCCA	4.1.6

ON statement	CCCA	4.1.4
ON SIZE ERROR option--changes in intermediate results	CCCA	4.1.6
OPEN statement failing for VSAM files (File Status 39)	None	4.1.4
OPEN statement failing for SAM files (File Status 39)	None	4.1.4
OTHERWISE clause changes	CCCA	4.1.6
PERFORM statement--changes in the VARYING/AFTER options	CCCA	4.1.6
Periods on paragraphs missing	CCCA	4.1.5
PROGRAM COLLATING SEQUENCE clause changes	CCCA	4.1.6
READ statement - redefined record keys in the KEY phrase	None	4.1.5
READ and RETURN statement changes--INTO phrase	None	4.1.6
READY TRACE and RESET TRACE Statements	CCCA	4.1.4
RECORD CONTAINS n CHARACTERS clause	None	4.1.5
REDEFINES clause in SD or FD entries	CCCA	4.1.5
REDEFINES clause with tables	None	4.1.5
Relation conditions	CCCA	4.1.5
REMARKS paragraph	CCCA	4.1.4
RENAMES clause - nonunique, nonqualified data names	None	4.1.5
Report Writer	Report Writer Precompiler	4.1.3.1
RERUN clause changes	None	4.1.6
RESERVE clause changes	CCCA	4.1.6
Reserved Word List changes	CCCA	4.1.6
SEARCH statement changes	CCCA	4.1.6
SEARCH or SEARCH ALL... WHEN with no imperative	none	4.1.4
Segmentation changes--PERFORM statement in independent segments	None	4.1.6
SELECT OPTIONAL clause changes	CCCA	4.1.6
SERVICE RELOAD	none	4.1.4
SPECIAL-NAMES paragraph changes	None	4.1.6
SORT special registers	None	4.1.6
SORT-OPTION IS clause	none	4.1.4
START ... USING KEY statement	CCCA	4.1.4

STRING statement - in PROGRAM COLLATING SEQUENCE clause	CCCA	4.1.6
Subscripts Out of Range--flagged at compile time	None	4.1.6
THEN as a statement connector	CCCA	4.1.4
TIME-OF-DAY special register	CCCA	4.1.4
TRANSFORM statement	CCCA	4.1.4
UNSTRING statement - in PROGRAM COLLATING SEQUENCE clause	CCCA	4.1.6
UNSTRING statement - coding with 'OR' and 'IS'	CCCA	4.1.5
UNSTRING statements--subscript evaluation changes	CCCA	4.1.6
UPSI switches	CCCA	4.1.6
USE AFTER STANDARD ERROR - GIVING option	CCCA	4.1.4
USE BEFORE STANDARD LABEL statement.	CCCA	4.1.4
VALUE clause - signed value in relation to the PICTURE clause	CCCA	4.1.5
VALUE clause - condition names	CCCA	4.1.6
WHEN-COMPILED special register	CCCA	4.1.6
WRITE AFTER POSITIONING statement	CCCA	4.1.6
Note:		
(1) This is a partial conversion.		

4.1.2 Use Conversion Aids to Convert Programs to COBOL 85 Standard

To help you estimate changes needed when upgrading to COBOL/VSE NOCMPR2 you can use any of the following:

- The DOS/VS COBOL MIGR compiler option
- The COBOL/VSE FLAGMIG and CMPR2 compiler options
- The COBOL and CICS Command Level Conversion Aid for VSE (CCCA)

A brief description of these conversion aids follows. See Appendix B, "Conversion Aids for Source Programs" in topic APPENDIX1.2 for additional information.

Subtopics:

4.1.2.1 DOS/VS COBOL MIGR Compiler Option

The DOS/VS COBOL MIGR compiler option flags most statements in a DOS/VS COBOL program that are not supported or are changed in COBOL/VSE with NOCMPR2. The MIGR compiler option allows you to analyze the conversion effort, and helps you identify required changes, without purchasing any conversion aids. Therefore, for each of your programs, even before conversion, you can get a good idea of how much conversion effort will be required.

"DOS/VS COBOL Conversion Aid--MIGR Compiler Option" in topic APPENDIX1.2.2 lists both the items flagged and not flagged by MIGR. A complete description of MIGR-flagged items is included in Appendix G of *IBM VS COBOL for DOS/VSE*.

4.1.2.2 COBOL/VSE CMPR2, FLAGMIG and NOCOMPILE Compiler Options

Another way you can identify DOS/VS COBOL and VS COBOL II Release 2 statements that are either not supported or changed in COBOL/VSE NOCMPR2, is to use the COBOL/VSE FLAGMIG compiler option together with the COBOL/VSE CMPR2 compiler option. By compiling existing application programs with the COBOL/VSE compiler, you can identify some of the source language that needs modification.

4.1.2.3 CCCA Program Offering

You can use the COBOL and CICS Command Level Conversion Aid for VSE (CCCA) Program Offering, Product Number 5785-CCC, to flag the affected elements and automatically convert them when possible. For details, see "COBOL and CICS Command Level Conversion Aid for VSE (CCCA)" in topic APPENDIX1.2.3.1 and the *COBOL and CICS Command Level Conversion Aid for VSE Installation and User's Guide*.

4.1.3 DOS/VS COBOL Language Elements No Longer Implemented

The following DOS/VS COBOL language elements are either not supported by COBOL/VSE or supported by use of a program offering such as:

- Report Writer
- ISAM file handling
- DAM file handling
- Segmentation - Conversion Action

In COBOL/VSE, support for most of the COBOL 68 Standard language elements that are no longer in COBOL 74 or COBOL 85 Standards, have been removed.

There are also miscellaneous DOS/VS COBOL language items that are not implemented in COBOL/VSE.

The conversion actions you can perform, and the language elements affected are documented in the following sections. There is a brief description of each item, plus conversion suggestions, and, where helpful, coding examples.

Subtopics:

4.1.3.1 Report Writer

The Report Writer feature is supported through use of the Report Writer Precompiler Program Offering. In order for existing Report Writer code to work with COBOL/VSE, you have three alternatives.

Keep Existing Report Writer Code and use the Report Writer Precompiler: When you recompile existing Report Writer applications (or newly- written applications) with the Report Writer Precompiler, and use the output as input to the COBOL/VSE compiler, your Report Writer applications can run above the 16-megabyte line. Through COBOL/VSE, you can also extend their processing capabilities.

This method requires the use of both the Report Writer Precompiler and the COBOL/VSE compiler.

Convert Existing Report Writer Code using the Report Writer Precompiler: If you permanently convert Report Writer code to non-Report Writer code, you can stop using the Report Writer Precompiler and just use the COBOL/VSE compiler. However, this might produce hard-to-maintain COBOL code.

When converting Report Writer code to non-Report Writer code, the precompiler generates variable names and paragraph names. These names might not be meaningful, and thus hard to identify when attempting to make changes to the program after the conversion. You can change the names to be meaningful, but this might be difficult and time consuming.

Run Existing DOS/VS COBOL-Compiled Report Writer Programs under LE/VSE: You can run existing DOS/VS COBOL Report Writer applications using LE/VSE without compiling with COBOL/VSE. For details on running existing DOS/VS COBOL programs using the LE/VSE run-time library, see Chapter 5, "Moving from the DOS/VS COBOL Run Time to LE/VSE" in topic 3.2. To compile DOS/VS COBOL applications with Report Writer statements, you must continue to use the DOS/VS COBOL compiler.

DOS/VS COBOL Report Writer programs will not run above the 16-megabyte line.

Report Writer Language Items Affected: The Report Writer language items no longer accepted by COBOL/VSE are:

```
GENERATE statement
INITIATE statement
LINE-COUNTER special register
Nonnumeric literal IS mnemonic-name
PAGE-COUNTER special register
PRINT-SWITCH special register
REPORT clause of FD entry
REPORT SECTION
TERMINATE statement
USE BEFORE REPORTING declarative
```

The Report Writer Precompiler is described in Appendix B, "Conversion Aids for Source Programs" in topic APPENDIX1.2.

4.1.3.2 ISAM File Handling

COBOL/VSE does not support the processing of ISAM files.

Subtopics:

- 4.1.3.2.1 ISAM File Handling Language Items Affected
- 4.1.3.2.2 ISAM File Handling Conversion Actions

4.1.3.2.1 ISAM File Handling Language Items Affected

The ISAM language items no longer accepted by COBOL/VSE are:

```
APPLY CORE-INDEX
APPLY REORG-CRITERIA
File declarations for ISAM files
NOMINAL KEY clause
Organization parameter I
TRACK-AREA clause
USING KEY clause of START statement
```

4.1.3.2.2 ISAM File Handling Conversion Actions

You should convert any ISAM files to Virtual Storage Access Method/Keyed Sequential Data Set (VSAM/KSDS) files. For such file processing programs, convert ISAM files to VSAM/KSDS manually. The IDCAMS REPRO facility will perform this conversion unless the file has a hardware dependency.

The COBOL and CICS Command Level Conversion Aid for VSE (CCCA) program offering can automatically convert the file definition and I/O statements from your ISAM COBOL language to VSAM/KSDS COBOL language. The CCCA conversion aid is described in Appendix B, "Conversion Aids for Source Programs" in topic APPENDIX1.2.

If the design of your application makes it impossible to convert to VSAM, you can restructure the application to separate the ISAM statements into an I/O program that can be compiled by the DOS/VS COBOL compiler. You can then separate the rest of the application logic into programs that can be upgraded to COBOL/VSE.

You can then run your application in a mixed environment using LE/VSE. (In a mixed environment, you can run both DOS/VS COBOL object programs and COBOL/VSE object programs using LE/VSE. For details, see Chapter 5, "Moving from the DOS/VS COBOL Run Time to LE/VSE" in topic 3.2.)

Note: This method is recommended only as long as IBM provides service and support for DOS/VS COBOL. Eventually, you should rewrite your programs to eliminate the dependency on DOS/VS COBOL and ISAM.

4.1.3.3 DAM File Handling

COBOL/VSE does not support the processing of DAM files.

Subtopics:

4.1.3.3.1 DAM File Handling Language Items Affected

The DAM language items no longer accepted by COBOL/VSE are:

ACTUAL KEY clause
APPLY RECORD-OVERFLOW
File declarations for DAM files
Organization parameters D, R, W
SEEK statement
TRACK-LIMIT clause

4.1.3.3.2 DAM File Handling Conversion Actions

You should convert any DAM files to Virtual Storage Access Method/Relative Record Data Set (VSAM/RRDS) files. For such file processing programs, convert DAM files to VSAM/RRDS manually. The IDCAMS REPRO facility will perform this conversion unless the file has a hardware dependency.

The COBOL and CICS Command Level Conversion Aid for VSE (CCCA) program offering can automatically convert your DAM COBOL language to VSAM/RRDS COBOL language. You must provide the key algorithm. The CCCA conversion aid is described in Appendix B, "Conversion Aids for Source Programs" in topic APPENDIX1.2.

If the design of your application makes it impossible to convert to VSAM, you can restructure the application to separate the DAM statements into an I/O program that can be compiled by the DOS/VS COBOL compiler. You can

then separate the rest of the application logic into programs that can be upgraded to COBOL/VSE.

You can then run your application in a mixed environment using LE/VSE. (In a mixed environment, you can run both DOS/VS COBOL object programs and COBOL/VSE object programs using LE/VSE. For details, see Chapter 5, "Moving from the DOS/VS COBOL Run Time to LE/VSE" in topic 3.2.)

Note: This method is recommend only as long as IBM provides service and support for DOS/VS COBOL. Eventually, you should rewrite your programs to eliminate the dependency on DOS/VS COBOL and DAM.

4.1.3.4 Segmentation - Conversion Actions

No programming changes are necessary.

If you have existing applications which contain segment priority numbers on section names or the SEGMENT-LIMIT clause coded in the OBJECT-COMPUTER paragraph, COBOL/VSE accepts the segmentation language but does not perform overlay. Therefore, storage allocation will not improve.

4.1.4 Miscellaneous Unsupported DOS/VS COBOL Language Elements

COBOL/VSE does not support the following DOS/VS COBOL language elements. When upgrading to COBOL/VSE, you must either remove or change these items as indicated in the following descriptions:

ASSIGN TO *integer system-name*
DOS/VS COBOL accepts the ASSIGN TO *integer system-name* clause.

To use this clause under COBOL/VSE, you must remove the integer.

ASSIGN ... FOR MULTIPLE REEL/UNIT
DOS/VS COBOL accepts the ASSIGN ... FOR MULTIPLE REEL/UNIT option, and treats it as documentation.

COBOL/VSE does not support this option.

COM-REG Special Register
DOS/VS COBOL accepts the COM-REG special register to allow access to the VSE communication region. COBOL/VSE does not support this option. Therefore you must remove any occurrences of the COM-REG special register.

CURRENT-DATE Special Register
DOS/VS COBOL accepts the CURRENT-DATE special register. It is valid only as the sending field in a MOVE statement. CURRENT-DATE has the 8-byte alphanumeric format:

MM/DD/YY (month, day, year)

COBOL/VSE supports the DATE special register. It is valid only as the sending field in an ACCEPT statement. (It is not valid

under CICS.) DATE has the 6-byte alphanumeric format:

```
      YYMMDD  (year, month, day)
```

Therefore, a DOS/VS COBOL program with statements similar to the following:

```
  77  DATE-IN-PROGRAM  PICTURE X(8)
           .
           .
           .
      MOVE CURRENT-DATE TO DATE-IN-PROGRAM.
```

must be changed. An example of one way to change it is as follows:

```
  01  DATE-IN-PROGRAM.
      02 MONTH-OF-YEAR     PIC X(02).
      02 FILLER            PIC X(01)  VALUE "/".
      02 DAY-OF-MONTH      PIC X(02).
      02 FILLER            PIC X(01)  VALUE "/".
      02 YEAR              PIC X(02).


  01  ACCEPT-DATE.
      02 YEAR              PIC X(02).
      02 MONTH-OF-YEAR     PIC X(02).
      02 DAY-OF-MONTH      PIC X(02).
           .
           .
           .


      ACCEPT ACCEPT-DATE FROM DATE.
      MOVE  CORRESPONDING ACCEPT-DATE TO DATE-IN-PROGRAM
```

EXAMINE Statement

DOS/VS COBOL accepts the EXAMINE statement.

COBOL/VSE does not accept the EXAMINE statement, treating is as an error.

Therefore, if your DOS/VS COBOL program contains coding similar to the following:

```
           .
           .
           .
      EXAMINE DATA-LENGTH TALLYING UNTIL FIRST " "
           .
           .
           .
```

Replace it in COBOL/VSE with:

```
           .
           .
           .
      MOVE 0 TO TALLY
      INSPECT DATA-LENGTH TALLYING TALLY FOR CHARACTERS BEFORE " "
           .
           .
           .
```

You can continue to use the TALLY special register wherever you can specify a WORKING-STORAGE elementary data item of integer value.

EXHIBIT Statement

DOS/VS COBOL accepts the EXHIBIT statement.

COBOL/VSE does not accept the EXHIBIT statement, treating it as an error.

In COBOL/VSE, you can use DISPLAY statements to replace EXHIBIT statements. However, the DISPLAY statement does not perform all the functions of the EXHIBIT statement.

The following changes are needed:

EXHIBIT NAMED: You can replace the EXHIBIT NAMED statement directly with a DISPLAY statement:

DOS/VS COBOL

```
          .
          .
          .
  WORKING-STORAGE SECTION.
  77  DAT-1 PIC X(8).
  77  DAT-2 PIC X(8).
          .
          .
          .
      EXHIBIT NAMED DAT-1 DAT-2
          .
          .
          .
```

COBOL/VSE

```
          .
          .
          .
  WORKING-STORAGE SECTION.
  77  DAT-1 PIC X(8).
  77  DAT-2 PIC X(8).
          .
          .
          .
      DISPLAY "DAT-1 = " DAT-1
          "DAT-2 = " DAT-2
          .
          .
          .
```

EXHIBIT CHANGED: You can replace the EXHIBIT CHANGED statement with IF and DISPLAY statements, as follows:

1. Specify an IF statement to discover if the new value of the data item is different from the old

2. Specify a DISPLAY statement as the *statement-1* of the IF statement

This displays the value of the specified data item(s) only if the new value is different from the old:

DOS/VS COBOL

```
          .
          .
          .
  WORKING-STORAGE SECTION.
  77  DAT-1 PIC X(8).
  77  DAT-2 PIC X(8).
          .
          .
```

COBOL/VSE

```
          .
          .
          .
  WORKING-STORAGE SECTION.
  77  DAT-1 PIC X(8).
  77  DAT-2 PIC X(8).
  77  DAT1-CMP PIC X(8).
  77  DAT2-CMP PIC X(8).
```

```
          .                                  .
     EXHIBIT CHANGED DAT-1 DAT-2             .
          .                                  .
          .                        IF DAT-1 NOT EQUAL TO DAT1-CMP
          .                          DISPLAY DAT-1
                                   END-IF
                                    IF DAT-2 NOT EQUAL TO DAT2-CMP
                                      DISPLAY DAT-2
                                    END-IF
                                    MOVE DAT-1 TO DAT1-CMP
                                    MOVE DAT-2 TO DAT2-CMP
                                           .
                                           .
                                           .
```

EXHIBIT CHANGED NAMED: You can replace the EXHIBIT CHANGED NAMED statement with IF and DISPLAY statements, as follows:

1. Specify an IF statement to discover if the new value of the data item is different from the old

2. Specify a DISPLAY statement as the *statement-1* of the IF statement

This displays the value of the specified data item(s) only if the new value is different from the old:

DOS/VS COBOL

```
          .
          .
          .
  WORKING-STORAGE SECTION.
  77  DAT-1 PIC X(8).
  77  DAT-2 PIC X(8).
          .
          .
          .
      EXHIBIT CHANGED NAMED
          DAT-1 DAT-2
          .
          .
          .
          .
```

COBOL/VSE

```
                                  .
                                  .
                                  .
WORKING-STORAGE SECTION.
77  DAT-1    PIC X(8).
77  DAT-2    PIC X(8).
77  DAT1-CMP PIC X(8).
77  DAT2-CMP PIC X(8).
                                  .
                                  .
                                  .
 IF DAT-1 NOT EQUAL TO DAT1-CMP
     DISPLAY "DAT-1 = " DAT-1
 END-IF
 IF DAT-2 NOT EQUAL TO DAT2-CMP
     DISPLAY "DAT-2 = " DAT-2
 END-IF
 MOVE DAT-1 TO DAT1-CMP
 MOVE DAT-2 TO DAT2-CMP
                                  .
                                  .
                                  .
```

FILE-LIMIT Clause of the FILE-CONTROL Paragraph

DOS/VS COBOL accepts the FILE-LIMIT clause and treats it as a comment; COBOL/VSE does not. Therefore, you must remove any occurrences of the FILE-LIMIT clause.

GIVING Option of USE AFTER STANDARD ERROR Declarative

In DOS/VS COBOL you can specify the GIVING option of the USE AFTER STANDARD ERROR declarative. COBOL/VSE does not support this option. Therefore, you must remove any occurrences of the GIVING option of the USE AFTER STANDARD ERROR declarative.

Use the FILE-CONTROL FILE STATUS clause to replace the GIVING option. The FILE STATUS clause gives you information after each I/O request, rather than only after an error occurs.

LABEL RECORDS is data-name for NON-SEQUENTIAL Files

DOS/VS COBOL accepts the LABEL RECORDS is data-name for non-sequential files. VS COBOL II does not.

Therefore, you must remove any occurrences of the LABEL RECORDS is data-name in non-sequential files.

NOTE Statement

DOS/VS COBOL accepts the NOTE statement. COBOL/VSE does not. Therefore, for COBOL/VSE delete all NOTE statements and use comment lines instead for the entire NOTE paragraph.

NSTD-REELS Special Register

DOS/VS COBOL accepts the NSTD-REELS special register.

COBOL/VSE does not accept the NSTD-REELS special register, replacing it with an operator prompt. Therefore, you must remove any references to the NSTD-REELS special register.

NUMERIC Class Tests for Group Items

DOS/VS COBOL accepts NUMERIC class tests for group items, but COBOL/VSE does not.

Therefore, you must remove any NUMERIC class test for group items.

ON Statement

DOS/VS COBOL accepts the ON statement. COBOL/VSE does not.

The ON statement allows selective execution of statements it contains. Similar functions are provided in COBOL/VSE by the EVALUATE statement and the IF statement.

OPEN Statement Failing for SAM Files - (File Status 39)

In DOS/VS COBOL the file record length for SAM files does not need to match your COBOL program. In COBOL/VSE, if the following do not match, an OPEN statement in your program might not execute successfully:

- The fixed file attributes for a SAM ESDS from the VSAM catalog or the file label for a file
- The attributes specified for that file in the SELECT and FD statements of your COBOL program

Mismatches in the attributes for file organization, record format (fixed or variable), the code set, or record length result in a file status code 39, and the OPEN statement fails.

To prevent common file status 39 problems, see Appendix J, "Preventing File Status 39 for SAM Files" in topic APPENDIX1.10.

OPEN Statement Failing for VSAM files - (File Status 39)

In DOS/VS COBOL the RECORDSIZE defined in your VSAM files associated with IDCAMS are not required to match your COBOL program. In COBOL/VSE they must match.

The following rules apply to VSAM ESDS, KSDS, RRDS, and VRRDS file definitions:

Figure 25. Rules for VSAM File Definitions

File Type	Rules
ESDS and KSDS VSAM	RECORDSIZE(avg,m) is specified where avg is the average size of the COBOL records, and is strictly less then m; m is greater than or equal to the maximum size of COBOL record.
RRDS VSAM	RECORDSIZE(n,n) is specified where n is greater than or equal to the maximum size of COBOL record.
VRRDS using KSDS VSAM	RECORDSIZE(avg,m) is specified where avg is the average size of the COBOL records, and is strictly less than m; m is greater than or equal to the maximum size of COBOL record + 4.

READY TRACE and RESET TRACE Statements

DOS/VS COBOL allows the READY TRACE and RESET TRACE statements. These statements are not supported in COBOL/VSE and are flagged as errors.

The COBOL/VSE USE FOR DEBUGGING ON ALL PROCEDURES declarative can perform functions similar to READY TRACE and RESET TRACE. For example:

```
          .
          .
  ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
    SOURCE-COMPUTER. IBM-370 WITH DEBUGGING MODE.
          .
          .
  DATA DIVISION.
          .
          .
    WORKING-STORAGE SECTION.
    01 TRACE-SWITCH        PIC 9 VALUE ZERO.
       88  TRACE-OFF             VALUE 0.
       88  TRACE-ON              VALUE 1.
          .
          .
  PROCEDURE DIVISION.
    DECLARATIVES.
    COBOL-II-DEBUG SECTION.
      USE FOR DEBUGGING ON ALL PROCEDURES.
    COBOL-II-DEBUG-PARA.
      IF TRACE-ON
          DISPLAY DEBUG-NAME
      END-IF.
    END DECLARATIVES.
    MAIN-PROCESSING SECTION.
          .
          .
    PARAGRAPH-3.
          .
          .
      MOVE 1 TO TRACE-SWITCH.
    PARAGRAPH-4.
```

```
           .
           .
       PARAGRAPH-6.
           .
           .
         MOVE 0 TO TRACE-SWITCH.
       PARAGRAPH-7.
```

where DEBUG-NAME is a field of the DEBUG-ITEM special register that displays the procedure-name causing execution of the debugging procedure. (In this example, the object program displays the names of procedures PARAGRAPH-4 through PARAGRAPH-6 as control reaches each procedure within the range.)

At run time, you must specify PARM=DEBUG in your EXEC statement to activate this debugging procedure. In this way, you have no need to recompile the program to activate or deactivate the debugging declarative.

REMARKS Paragraph

DOS/VS COBOL accepts the REMARKS paragraph. COBOL/VSE does not. As a replacement, use comment lines beginning with an * in column 7.

SEARCH or SEARCH ALL... WHEN with No Imperative

DOS/VS COBOL supports the SEARCH or SEARCH ALL... WHEN with no imperative. COBOL/VSE does not.

SERVICE RELOAD

DOS/VS COBOL supports the SERVICE RELOAD option to allow reloading of BLL cells. COBOL/VSE does not support this option because it automatically reloads BLL cells.

SORT-OPTION IS Clause

DOS/VS COBOL supports the SORT-OPTION IS clause. COBOL/VSE does not. The equivalent function is provided under COBOL/VSE by the SORT-CONTROL special register, in combination with the DFSORT/VSE OPTION control statement.

START...USING KEY Statement

DOS/VS COBOL allows the START statement with the USING KEY option. COBOL/VSE does not. Instead, for COBOL/VSE, you can specify the START statement with the KEY IS option.

THEN as a Statement Connector

DOS/VS COBOL accepts the use of THEN as a statement connector.

The following example shows the DOS/VS COBOL usage:

```
MOVE A TO B THEN ADD C TO D
```

COBOL/VSE does not support the use of THEN as a statement connector. Therefore, in COBOL/VSE change it to:

```
MOVE A TO B
ADD C TO D
```

TIME-OF-DAY Special Register

DOS/VS COBOL supports the TIME-OF-DAY special register. It is valid only as the sending field in a MOVE statement. TIME-OF-DAY has the following 6-byte external decimal format:

```
     HHMMSS (hour, minute, second)
```

COBOL/VSE supports the TIME special register from the COBOL 74 Standard. It is valid only as the sending field in an ACCEPT statement. TIME has the 8-byte external decimal format:

```
     HHMMSSCC (hour, minute, second, hundredth-of-second)
```

Therefore, you must change a DOS/VS COBOL program with statements similar to the following:

```
  77  TIME-IN-PROGRAM   PICTURE  X(6).
          .
          .
          .
      MOVE TIME-OF-DAY TO TIME-IN-PROGRAM.
```

An example of one way to change it is as follows:

```
  01  ACCEPT-TIME.
      02  TIME-IN-PROGRAM  PIC X(06).
      02  FILLER           PIC X(02).
          .
          .
          .
      ACCEPT ACCEPT-TIME FROM TIME.
```

Note: Neither TIME-OF-DAY nor TIME are valid under CICS.

TRANSFORM Statement

DOS/VS COBOL supports the TRANSFORM statement. COBOL/VSE supports the INSPECT statement. Therefore, any TRANSFORM statements in your DOS/VS COBOL program must be replaced by INSPECT CONVERTING statements.

For example, in the following DOS/VS COBOL TRANSFORM statement:

```
  77  DATA-T     PICTURE X(9) VALUE "ABCXYZCCC"
          .
          .
      TRANSFORM DATA-T FROM "ABC" TO "CAT"
```

TRANSFORM evaluates each character, changing each A to C, each B to A, and each C to T.

After the TRANSFORM statement is executed, DATA-T contains "CATXYZTTT".

For example, in the following INSPECT CONVERTING statement (valid only in COBOL/VSE):

```
  77  DATA-T     PICTURE X(9) VALUE "ABCXYZCCC"
         .
         .
      INSPECT DATA-T
          CONVERTING "ABC" TO "CAT"
```

INSPECT CONVERTING evaluates each character just as TRANSFORM

does, changing each A to C, each B to A, and each C to T.

After the INSPECT CONVERTING statement is executed. DATA-T contains "CATXYZTTT".

USE BEFORE STANDARD LABEL

DOS/VS COBOL accepts the USE BEFORE STANDARD LABEL statement. COBOL/VSE does not support this statement.

Therefore, you must remove any occurrences of the USE BEFORE STANDARD LABEL statement. COBOL/VSE does not support nonstandard labels, so you cannot process nonstandard labeled files with COBOL/VSE.

4.1.5 Undocumented DOS/VS COBOL Extensions

The following examples consist primarily of COBOL statements that are not flagged by the MIGR option. These examples are accepted by the DOS/VS COBOL compiler; some are not accepted by COBOL/VSE.

Because they are undocumented extensions to DOS/VS COBOL, they are not considered to be valid DOS/VS COBOL code.

In a few instances, examples are included of statements that will generate a compiler message under DOS/VS COBOL even though they will compile under COBOL/VSE.

This section shows examples of undocumented extensions; it is not definitive.

ACCEPT identifier FROM SYSIPT Statement

DOS/VS COBOL allows multiple logical files to be read from SYSIPT using the ACCEPT identifier FROM SYSIPT statement. After end-of-file is detected on SYSIPT, a subsequent ACCEPT identifier FROM SYSIPT statement will read the next card image from SYSIPT.

COBOL/VSE does not support multiple SYSIPT input files. After end-of-file is detected on SYSIPT, a subsequent ACCEPT identifier FROM SYSIPT statement will leave *identifier* unchanged.

BLANK WHEN ZERO Clause and Asterisk (*) Replace

In DOS/VS COBOL, if you specify the BLANK WHEN ZERO clause and the asterisk (*) as a zero suppression symbol for the same entry, zero suppression replaces BLANK WHEN ZERO.

COBOL/VSE does not accept these two language elements when they are specified for the same data description entry. Therefore COBOL/VSE must not contain instances of both the clause and the symbol in one program.

If you have specified both the BLANK WHEN ZERO clause and the asterisk as a zero suppression symbol in your DOS/VS COBOL programs, to get the same behavior in COBOL/VSE, remove the BLANK WHEN ZERO clause.

Index Names

DOS/VS COBOL allows the use of qualified index names. In COBOL/VSE, index names must be unique.

MOVE Statement

Although the COBOL/VSE TRUNC(OPT) compiler option is recommended for compatibility with the DOS/VS COBOL NOTRUNC compiler option, you might receive different results involving moves of fullword binary items (USAGE COMP with Picture 9(5) thru Picture 9(9)).

For example:

```
  WORKING-STORAGE SECTION.
    01 WK1 USAGE COMP-4 PIC S9(9).
           .
           .
           .
  PROCEDURE DIVISION.
           .
           .
           .
    MOVE 1234567890 to WK1
    DISPLAY WK1.
    GOBACK.
```

For example, the results are as follows when compiled with the following compiler options:

	DOS/VS COBOL NOTRUNC	**COBOL/VSE TRUNC(OPT)**
Binary Value	x'499602D2'	x'0DFB38D2'
Display Value	234567890	234567890

For DOS/VS COBOL, the binary value contained in the binary data item is not the same as the display value. The display value is based on the number of digits in the PICTURE clause and the binary value is based on the size of the binary data item, in this case, 4 bytes. The actual value of the binary data item in decimal digits is 1234567890.

For COBOL/VSE, the binary value and the display value are equal because the truncation that occurred was based on the number of digits in the PICTURE clause.

This situation is flagged by MIGR in DOS/VS COBOL and by TRUNC(OPT) in COBOL/VSE.

MOVE CORRESPONDING Statement

COBOL/VSE does not allow more than one receiving field. Therefore, you must change the following DOS/VS COBOL statement:

```
MOVE CORRESPONDING GROUP-ITEM-A TO GROUP-ITEM-B GROUP-ITEM-C
```

to two COBOL/VSE MOVE CORRESPONDING statements:

```
MOVE CORRESPONDING GROUP-ITEM-A TO GROUP-ITEM-B
```

```
MOVE CORRESPONDING GROUP-ITEM-A TO GROUP-ITEM-C
```

OCCURS Clause

DOS/VS COBOL allows nonstandard order for phrases following the OCCURS clause.

For example, the following code sequence is allowed in DOS/VS COBOL, but generates an E-level and an S-level error message in COBOL/VSE:

```
01  D   PIC 999.
01  A.
    02  B OCCURS 1 TO 200 TIMES
          ASCENDING KEY C
          DEPENDING ON D
          INDEXED BY H.
    03  C PIC 99.
```

Therefore, you must reorder the phrases following the OCCURS clause.

In COBOL/VSE you should code OCCURS clause similar to the following example:

```
01 TBL.
  02 TBL-1 OCCURS 1 TO 200 TIMES
     DEPENDING ON N
     ASCENDING C
     INDEXED BY D.
  03 C PIC 9(5).
77 N PIC 9(3).
```

Periods on Paragraphs Missing

Releases prior to Release 3.0 of DOS/VS COBOL accepted paragraph names not followed by a period. For Release 3.0 of DOS/VS COBOL, a paragraph name without a period generates a W-level error message.

COBOL/VSE generates an E-level error message.

Program-ID names, NonUnique

DOS/VS COBOL allows a data-name or paragraph-name to be the same as the program-ID. COBOL/VSE requires the program-ID to be unique.

READ Statement

DOS/VS COBOL accepts implicitly or explicitly redefined record keys in the KEY phrase of the READ statement.

COBOL/VSE accepts only the names of the data items that are specified as record keys in the SELECT clause for the file being read.

RECORD CONTAINS n CHARACTERS Clause

In variation with the COBOL 74 Standard, the RECORD CONTAINS n CHARACTERS clause of a DOS/VS COBOL program will be replaced if an OCCURS DEPENDING ON clause is specified in the FD, and will produce a file containing **variable**-length records instead of **fixed**-length records.

Under COBOL/VSE, the RECORD CONTAINS n CHARACTERS clause

produces a file containing **fixed**-length records.

REDEFINES Clause in SD or FD Entries

Releases prior to Release 3.0 of DOS/VS COBOL accept a REDEFINES clause in a level-01 SD or FD. COBOL/VSE and Release 3.0 of DOS/VS COBOL do not.

For example, the following code sequence is invalid:

```
SD ...
01    SORT-REC-HEADER.
      05  SORT-KEY          PIC X(20).
      05  SORT-HEADER-INFO  PIC X(40).
      05  FILLER            PIC X(20).
01    SORT-REC-DETAIL REDEFINES SORT-REC-HEADER.
      05  FILLER            PIC X(20).
      05  SORT-DETAIL-INFO  PIC X(60).
```

To get similar function in COBOL/VSE, delete the REDEFINES clause.

REDEFINES Clause with Tables

DOS/VS COBOL allows you to specify tables within the REDEFINES clause. COBOL/VSE does not. For example, the following code generates a W-level warning message with DOS/VS COBOL (RC=4) and a S-level error message with COBOL/VSE (RC=12):

```
01 E.
    03 F OCCURS 10.
       05 G PIC X.
    03 I REDEFINES F PIC X.
```

Relation Conditions

The following shows the valid coding for operators in relation conditions:

Prior to DOS/VS COBOL R3.0	**COBOL/VSE**
= TO	= or EQUAL TO
> THAN	> or GREATER THAN
< THAN	< or LESS THAN

RENAMES Clause

No MIGR message will be issued if the RENAMES clause in your DOS/VS COBOL program references a nonunique, nonqualified data name. However, COBOL/VSE does not support the use of nonunique, nonqualified data names.

UNSTRING Statement

Under DOS/VS COBOL, your program will not generate a compiler message if it contains an UNSTRING statement containing either of the following instances of invalid coding:

1. Lack of the required word "OR" between identifier-2 and identifier-3, as in:

```
UNSTRING A-FIELD DELIMITED BY '-'  ','
    INTO RECV-FIELD-1
```

```
                POINTER PTR-FIELD.
```

2. Presence of the extraneous word "IS" in specifying a pointer, as in:

```
          UNSTRING A-FIELD DELIMITED BY '-' OR ','
              INTO RECV-FIELD-2
              POINTER IS PTR-FIELD.
```

COBOL/VSE will issue a message if an UNSTRING statement containing either of these errors is encountered.

VALUE clause

In DOS/VS COBOL, the VALUE clause can be signed if the PICTURE clause is unsigned. In COBOL/VSE, the VALUE clause must match the PICTURE clause and the sign must be removed.

4.1.6 Language Elements Changed from DOS/VS COBOL

In conforming to the COBOL 85 Standard, the following DOS/VS COBOL language elements are changed in COBOL/VSE. For some elements, the syntax of the language is different. For others, the language syntax is unchanged, but the execution results can be different.

For each element listed, there is a brief description pointing out the differences in results and what actions to take. Where it is helpful, clarifying coding examples are also given.

ALPHABETIC Class Changes

In DOS/VS COBOL, only uppercase letters and the space character are considered to be ALPHABETIC.

In COBOL/VSE uppercase letters, lowercase letters, and the space character are considered to be ALPHABETIC.

If your DOS/VS COBOL program uses the ALPHABETIC class test, and the data tested consists of mixed uppercase and lowercase letters, there can be differences in execution results. In such cases, you can ensure identical results by substituting the COBOL/VSE ALPHABETIC-UPPER class test for the DOS/VS COBOL ALPHABETIC test.

APPLY WRITE-ONLY Clause Restrictions Removed

In DOS/VS COBOL, when using the APPLY WRITE-ONLY clause, all WRITE statements must have FROM options. In addition, none of the subfields of the associated records can be referred to by procedure statements or be the object of the DEPENDING ON option in an OCCURS clause.

In COBOL/VSE these restrictions do not apply.

Arithmetic Statement Changes

COBOL/VSE supports the following arithmetic items with enhanced accuracy:

- Definitions of floating-point data items
- Use of floating-point literals
- Use of exponentiation

Therefore, for arithmetic statements that contain these items, COBOL/VSE might provide more accurate results than DOS/VS COBOL.

ASSIGN Clause Changes

COBOL/VSE supports only the following format of the ASSIGN clause:

```
ASSIGN TO assignment-name
```

Where *assignment-name* can have the following forms:

```
SAM Files                     [[SYSnnn-[mfdc-]]S-]name
VSAM Sequential Files         [label-][AS-]name
VSAM Indexed or Relative Files [label-]name
```

If your DOS/VS COBOL program uses other formats of the ASSIGN clause, or other forms of the *assignment-name*, you must change it to conform to the COBOL/VSE format.

B Symbol in PICTURE Clause--Changes in Evaluation

DOS/VS COBOL accepts the PICTURE symbols A and B in definitions for alphabetic items.

COBOL/VSE accepts only the PICTURE symbol A. (A PICTURE that contains both symbols A and B defines an alphanumeric edited item.)

This can cause execution differences between DOS/VS COBOL and COBOL/VSE for evaluations of:

- The class test
- The STRING statement

BLOCK CONTAINS Clause - Changes for SAM ESDS Files

In DOS/VS COBOL and COBOL/VSE with CMPR2, the block size of a SAM ESDS file is determined from the BLOCK CONTAINS clause (unless BLOCK CONTAINS 0 is specified), regardless of whether or not the file has been previously implicitly or explicitly defined to VSE/VSAM.

In COBOL/VSE with NOCMPR2, the block size of a previously implicitly or explicitly defined SAM ESDS file is determined from the VSE/VSAM catalog, not from the value specified in the BLOCK CONTAINS clause.

If your DOS/VS COBOL program depends upon the block size specified in the BLOCK CONTAINS clause overriding the block size in the catalog entry of a previously defined SAM ESDS file, you can compile the program under COBOL/VSE specifying the CMPR2 option.

CALL Statement Changes

DOS/VS COBOL accepts procedure names and file names in the USING option of the CALL statement.

COBOL/VSE CALL statements do not accept procedure-names and only accept SAM file names in the USING OPTION. Therefore, you must remove the procedure-names and make sure that file names used in the USING option of the CALL statement name SAM physical sequential files.

Note: You should not specify a sequential file name in the USING option of the CALL statement if the file is not open.

To convert DOS/VS COBOL programs that call assembler programs passing procedure-names, you will need to rewrite the assembler routines. In DOS/VS COBOL programs, assembler routines can be written to receive an address or a list of addresses from the paragraph name that was passed as a parameter. The assembler routines can then use this address to return to an alternative place in the main program if an error occurs.

In COBOL/VSE, code your assembler routines so that they return to the point of origin with an assigned number, such as zero. If an error occurs in the assembler program, this number can then be used to go to alternative places in the calling routine. For example:

DOS/VS COBOL:

```
CALL "ASMMOD" USING PARAMETER-1,
                    PARAGRAPH-1,
                    PARAGRAPH-2,
NEXT STATEMENT.
:
PARAGRAPH-1.
:
:
PARAGRAPH-2.
```

COBOL/VSE:

```
CALL "ASMMOD" USING PARAMETER-1,
                    PARAMETER-2.
IF PARAMETER-2 NOT = 0
   GOTO PARAGRAPH-1,
        PARAGRAPH-2,
        DEPENDING ON PARAMETER-2.
```

In this example, you would modify the assembler program (ASMMOD) so that it does not branch to an alternative location. Instead, it will pass back the number zero to the calling routine if there are no errors, and a nonzero return value if an error occurred. The nonzero value would be used to determine which paragraph in the COBOL program would handle the error condition.

Combined Abbreviated Relation Condition Changes

Three considerations affect combined abbreviated relation conditions:

- NOT and logical operator/relational operator evaluation
- Parenthesis evaluation
- Optional word IS

All are described in the following sections.

NOT and Logical Operator/Relational Operator Evaluation: DOS/VS COBOL with LANGLVL(1), accepts the use of NOT in combined abbreviated relation conditions as follows:

- When only the subject of the relation condition is implied, NOT is considered a logical operator, for example:

```
A = B AND NOT LESS THAN C OR D
```

is equivalent to:

```
((A = B) AND NOT (A < C) OR (A < D))
```

- When both the subject and the relational operator are implied, NOT is considered to be part of the relational operator, for example:

```
A > B AND NOT C
```

is equivalent to:

```
A > B AND A NOT > C
```

Both COBOL/VSE and DOS/VS COBOL, in combined abbreviated relation conditions, consider NOT to be:

- Part of the relational operator in the forms NOT GREATER THAN, NOT >, NOT LESS THAN, NOT <, NOT EQUAL TO, and NOT =, for example:

```
A = B AND NOT LESS THAN C OR D
```

is equivalent to:

```
((A = B) AND (A NOT < C) OR (A NOT < D))
```

- NOT in any other position is considered to be a logical operator (and thus results in a negated relation condition), for example:

```
A > B AND NOT C
```

is equivalent to:

```
A > B AND NOT A > C
```

To ensure that you get the execution results you want, expand all abbreviated combined conditions to their full unabbreviated forms.

Parenthesis Evaluation: DOS/VS COBOL accepts the use of parentheses within an abbreviated combined relational condition. (This is not a documented usage of DOS/VS COBOL)

COBOL/VSE supports most parenthesis usage as IBM extensions. However, there are some incompatibilities/differences:

- COBOL/VSE accepts the logical NOT operator preceding a left parenthesis. DOS/VS COBOL does not.

- Within the scope of an abbreviated combined relation

condition, COBOL/VSE does not support relational operators inside parentheses. For example:

```
A = B AND  ( < C OR D)
```

° Some incorrect usages of parenthesis in relation conditions are accepted by DOS/VS COBOL, but flagged as an error by COBOL/VSE. For example:

```
(A = 0 AND B) = 0
```

Optional Word IS: DOS/VS COBOL accepts the optional word IS immediately preceding objects within an abbreviated combined relation condition. For example:

```
A = B OR IS C AND IS D
```

COBOL/VSE does not accept this usage of the optional word IS.

Therefore, for COBOL/VSE, delete the word IS when used in this manner.

Note: COBOL/VSE does permit the use of the optional word IS as part of the relational operator in abbreviated combined relational conditions. For example:

```
A = B OR IS = C AND IS = D
```

COPY Statement Changes

DOS/VS COBOL with the LANGLVL(1) compiler option accepts COPY statements written to conform to COBOL 68 Standard. COBOL/VSE does not.

EXIT PROGRAM/GOBACK Statement Changes

In DOS/VS COBOL, when an EXIT PROGRAM or GOBACK statement is executed, if the end of range of a PERFORM statement within it has not been reached, the PERFORM statement remains in its uncompleted state.

In COBOL/VSE, when an EXIT PROGRAM or GOBACK statement is executed, the end of range of every PERFORM statement within it is considered to have been reached.

FILE STATUS Clause Changes

In DOS/VS COBOL, the FILE STATUS clause defines a single status key, which contains a status key value after every I/O operation is completed.

In COBOL/VSE, the FILE STATUS clause defines up to two status keys. (The second status key is only valid for VSAM files; it contains the VSAM return code, function code, and feedback code.)

In addition, for COBOL/VSE status key values have been changed:

° For SAM files, see Figure 26

- For VSAM files, see Figure 27

If your DOS/VS COBOL program uses status key values to determine the course of execution, you must modify the program to use the new status key values. For complete information on COBOL/VSE file status codes, see the *COBOL/VSE Language Reference*.

Figure 26. Status Key Values--SAM Files

DOS/VS COBOL Status Keys	COBOL/VSE Status Keys	Meaning
(undefined)	04	COBOL/VSE only: wrong length record Successful completion
(undefined)	05	Optional file not present Successful completion
(undefined)	07	NO REWIND/REEL/UNIT/FOR REMOVAL specified for OPEN or CLOSE, but file not on a reel/unit medium Successful completion
(undefined)	35	Nonoptional file not present
(undefined)	37	Device type conflict
(undefined)	39	Conflict of fixed file attributes; OPEN fails
(undefined)	96	No file identification (Label Information for this VSAM file)
00	00	Successful completion
00	92	A CLOSE REEL/UNIT was executed after end-of-file had been detected on the referenced file
10	10	At END (no next logical record) Successful completion
30	30	Permanent error
34	34	Permanent error File boundary violation
90	90	Other errors with no further information
92	38	OPEN attempted for file closed WITH LOCK
92	41	OPEN attempted for a file in OPEN mode
92	42	CLOSE attempted for a file not in OPEN mode
92	43	REWRITE attempted when last I/O statement was not READ
92	44	Attempt to rewrite a sequential file record with a record of a different size
92	46	Sequential READ attempted with no valid next record
92	47	READ attempted when file not in OPEN INPUT or I-O mode
92	48	WRITE attempted when file not in OPEN OUTPUT,

		I-O, or EXTEND mode
92	49	DELETE or REWRITE attempted when file not in OPEN I-O mode
92	92	Logic error

Figure 27. Status Key Values--VSAM Files

DOS/VS COBOL Status Keys	COBOL/VSE Status Keys	Meaning
(undefined)	14	On sequential READ for relative file, size of relative record number too large for relative key
00	00	Successful completion
00	04	Wrong length record Successful completion
00	05	Optional file not present Successful completion
02	02	Duplicate key, and DUPLICATES specified. Successful completion
10	10	At END (no next logical record) Successful completion
20	20	Invalid key for a VSAM indexed or relative file
21	21	Invalid key for a VSAM indexed or relative file; sequence error
22	22	Invalid key for a VSAM indexed or relative file; duplicate key and duplicates not allowed
23	23	Invalid key for a VSAM indexed or relative file; no record found
24	24	Invalid key for a VSAM indexed or relative file; attempt to write beyond file boundaries COBOL/VSE only: for a WRITE to a relative file, size of relative record number too large for relative key
30	30	Permanent error
90	37	Attempt to open a file not on a mass storage device
90	90	Other errors with no further information
91	91	VSAM password failure
92	41	OPEN attempted for a file in OPEN mode
92	42	CLOSE attempted for a file not in OPEN mode
92	43	REWRITE attempted when last I/O statement was not READ or DELETE
92	46	Sequential READ attempted with no valid next

		record
92	47	READ attempted when file not in OPEN INPUT or I-O mode
92	48	WRITE attempted when file not in OPEN OUTPUT, I-O, or EXTEND mode
92	49	DELETE or REWRITE attempted when file not in OPEN I-O mode
93	93	VSAM resource not available
93 96	35	Nonoptional file not present
94	94	Under CMPR2: No file position indicator for VSAM sequential request
95	39	Conflict of fixed file attributes; OPEN fails
95	95	Invalid or incomplete VSAM file information
96	96	No file identification (no Label Information for this VSAM file)
97	97	OPEN statement execution successful; file integrity verified

IF...OTHERWISE Statement Changes

DOS/VS COBOL allows IF statements of the nonstandard format:

```
IF condition THEN statement-1 OTHERWISE statement-2
```

COBOL/VSE allows only IF statements having the standard format:

```
IF condition THEN statement-1 ELSE statement-2
```

Therefore, DOS/VS COBOL programs containing nonstandard IF...OTHERWISE statements must be changed to standard IF...ELSE statements.

JUSTIFIED Clause Changes

Under DOS/VS COBOL with LANGLVL(1), if a JUSTIFIED clause is specified together with a VALUE clause for a data description entry, the initial data is right-justified. For example:

```
  77  DATA-1 PIC X(9) JUSTIFIED VALUE "FIRST".
```

results in "FIRST" occupying the five rightmost character positions of DATA-1:

```
  bbbbFIRST
```

In COBOL/VSE, the JUSTIFIED clause does not affect the initial placement of the data within the data item. If a VALUE and JUSTIFIED clause are both specified for an alphabetic or alphanumeric item, the initial value is left-justified within the data item. For example:

```
  77  DATA-1 PIC X(9) JUSTIFIED VALUE "FIRST".
```

results in "FIRST" occupying the five leftmost character positions of DATA-1:

```
  FIRSTbbbb
```

To achieve unchanged results in COBOL/VSE, you can specify the literal value as occupying all nine character positions of DATA-1. For example:

```
  77  DATA-1 PIC X(9) JUSTIFIED VALUE "    FIRST".
```

which right-justifies the value in DATA-1:

```
  bbbbFIRST
```

MOVE Statements and Comparisons--Scaling Changes

In DOS/VS COBOL with LANGLVL(1) if either the sending field in a MOVE statement or a field in a comparison is a scaled integer (that is, if the rightmost PICTURE symbols are the letter P) and the receiving field (or the field to be compared) is alphanumeric or numeric-edited, the trailing zeros (0) are shortened.

For example, after the following MOVE statement is executed:

```
      05  SEND-FIELD     PICTURE 999PPP VALUE 123000.
      05  RECEIVE-FIELD  PICTURE XXXXXX.
           .
           .
           .
           MOVE SEND-FIELD TO RECEIVE-FIELD.
```

RECEIVE-FIELD contains the value 123bbb (left-justified).

With COBOL/VSE, a MOVE statement transfers the trailing zeros, and a comparison includes them.

For example, after the following MOVE statement is executed:

```
      05  SEND-FIELD     PICTURE 999PPP VALUE 123000.
      05  RECEIVE-FIELD  PICTURE XXXXXX.
           .
           .
           .
           MOVE SEND-FIELD TO RECEIVE-FIELD.
```

RECEIVE-FIELD contains the value 123000.

MULTIPLE FILE TAPE Clause

Under DOS/VS COBOL, the MULTIPLE FILE TAPE clause is used to control the positioning of multiple files that share the same physical reel of tape. The MULTIPLE FILE TAPE clause is pertinent only when labels are omitted. It is treated as a comment for a tape that has standard labels.

In COBOL/VSE, the MULTIPLE FILE TAPE clause is syntax-checked, but it has no effect on the execution of the program. The positioning of a multiple file tape must be performed by the system using JCL.

OCCURS DEPENDING ON Clause - ASCENDING/DESCENDING KEY Option

DOS/VS COBOL accepts a variable-length key in the ASCENDING/DESCENDING KEY option of the OCCURS DEPENDING ON clauses as an IBM extension in DOS/VS COBOL.

In COBOL/VSE, the ASCENDING/DESCENDING KEY phrase cannot specify a variable-length key.

OCCURS DEPENDING ON Clause - Value for Receiving Items Changed

In DOS/VS COBOL, the current value of the OCCURS DEPENDING ON (ODO) object is always used for both sending and receiving items.

In COBOL/VSE, for sending items, the current value of the ODO object is used. For receiving items:

1. If a group item contains both the subject and object of an ODO, the maximum length of the item is used.

2. If a receiving item containing an ODO is followed, in the same record, by a nonsubordinate data item, the actual length of the receiving item is used.

When the maximum length is used, it is not necessary to initialize the ODO object before the table receives data. For items whose length or location depend on the value of the ODO object, you need to set the object of the OCCURS DEPENDING ON clause before using it in the USING phrase of a CALL statement. Under COBOL/VSE, for any group item that contains the subject and object of the OCCURS DEPENDING ON clause, you do not need to set the object for the item when it is used in the USING BY REFERENCE phrase of the CALL statement. This is true even if the group is described by rule number 2 above.

The following example illustrates these points:

```
01  TABLE-GROUP-1
    05  ODO-KEY-1 PIC 99.
    05  TABLE-1 PIC X(9)
        OCCURS 1 TO 50 TIMES DEPENDING ON ODO-KEY-1.
01  ANOTHER-GROUP.
    05  TABLE-GROUP-2.
        10  ODO-KEY-2 PIC 99.
        10  TABLE-2 PIC X(9)
            OCCURS 1 to 50 TIMES DEPENDING ON ODO-KEY-2.
    05  TRAILER-ITEM PIC X(200).
           .
           .
PROCEDURE DIVISION.
           .
           .
    MOVE SEND-ITEM-1 TO TABLE-GROUP-1
           .
           .
    MOVE ODO-KEY-X TO ODO-KEY-2
    MOVE SEND-ITEM-2 TO TABLE-GROUP-2.
```

When TABLE-GROUP-1 is a receiving item, COBOL/VSE moves the maximum number of character positions for it (450 bytes for TABLE-1 plus two bytes for ODO-KEY-1). Therefore, you need not initialize the length of TABLE-1 before moving the SEND-ITEM-1

data into the table.

However, a nonsubordinate TRAILER-ITEM follows TABLE-GROUP-2 in the record description. In this case, COBOL/VSE uses the actual value in ODO-KEY-2 to calculate the length of TABLE-GROUP-2, and you must set ODO-KEY-2 to its valid current length before moving the SEND-ITEM-2 data into the group receiving item.

ON SIZE ERROR Option--Changes in Intermediate Results

For DOS/VS COBOL, the SIZE ERROR option for the DIVIDE and MULTIPLY statements applies to both intermediate and final results.

For COBOL/VSE, the SIZE ERROR option for the DIVIDE and MULTIPLY statements applies only to final results.

Therefore, if your DOS/VS COBOL depends upon SIZE ERROR detection for intermediate results, you must change your program.

PERFORM Statement--Changes in the VARYING/AFTER Options

In DOS/VS COBOL, in a PERFORM statement with the VARYING/AFTER options, two actions take place when an inner condition tests as TRUE:

1. The identifier/index associated with the inner condition is set to its current FROM value.

2. The identifier/index associated with the outer condition is increased by its current BY value.

In COBOL/VSE in such a PERFORM statement, the following takes place when an inner condition tests as TRUE:

1. The identifier/index associated with the outer condition is increased by its current BY value.

2. The identifier/index associated with the inner condition is set to its current FROM value.

The following example illustrates the differences in execution:

```
          .
          .
      PERFORM ABC VARYING X FROM 1 BY 1 UNTIL X > 3
                  AFTER Y FROM X BY 1 UNTIL Y > 3
          .
          .
```

In DOS/VS COBOL, ABC is executed 8 times with the following values:

```
     X:  1  1  1  2  2  2  3  3
     Y:  1  2  3  1  2  3  2  3
```

In COBOL/VSE, ABC is executed 6 times with the following values:

```
X:  1  1  1  2  2  3
Y:  1  2  3  2  3  3
```

By using nested PERFORM statements, you could achieve the same processing results in COBOL/VSE as in DOS/VS COBOL, as follows:

```
      MOVE 1 TO X, Y, Z
      PERFORM EX-1 VARYING X FROM 1 BY 1 UNTIL X > 3
      .
      .


  EX-1.
      PERFORM ABC VARYING Y FROM Z BY 1 UNTIL Y > 3.
      MOVE X TO Z.


  ABC.
      .
      .
      .
```

PROGRAM COLLATING SEQUENCE Clause Changes

In DOS/VS COBOL, the collating sequence specified in the *alphabet-name* of the PROGRAM COLLATING SEQUENCE clause is applied to comparisons implicitly performed during execution of INSPECT, STRING, and UNSTRING statements.

In COBOL/VSE, the collating sequence specified in *alphabet-name* is no longer used for these implicit comparisons.

READ and RETURN Statement Changes--INTO Phrase

When the sending field is chosen for the move associated with a READ or RETURN...INTO identifier statement, DOS/VS COBOL and COBOL/VSE can select different records from under the FD or SD to use as the sending field.

This only affects implicit elementary MOVEs, when the record description contains a PICTURE clause.

RERUN Clause Changes

The RERUN clause for COBOL/VSE has two formats:

- For SAM and VSAM files:
  RERUN ON *assignment-name*[EVERY]*integer* RECORDS [OF] *file-name*

- For Sort/Merge files:
  RERUN ON *assignment-name*, where *assignment-name* has the same format as *assignment-name* for SAM files.

In DOS/VS COBOL, only one RERUN clause may specify a DASD checkpoint file. In COBOL/VSE, you may have multiple DASD checkpoint files.

In COBOL/VSE, if checkpoint records are to be written to a tape file during a Sort/Merge operation, the logical unit number

specified in the *assignment-name* must be SYS000. If checkpoint records are to be written to a disk file during a Sort/Merge operation, the system-name specified in the *assignment-name* must be SORTCKP.

Magnetic tape files used for checkpoint files are always unlabeled.

RESERVE Clause Changes

The following formats of the FILE CONTROL paragraph RESERVE clause are supported:

DOS/VS COBOL	**COBOL/VSE**
RESERVE NO ALTERNATE AREA	
RESERVE NO ALTERNATE AREAS	
RESERVE integer ALTERNATE AREA	
RESERVE integer ALTERNATE AREAS	
RESERVE integer AREA	RESERVE integer AREA
RESERVE integer AREAS	RESERVE integer AREAS

If your DOS/VS COBOL program uses either the RESERVE integer ALTERNATE AREA or the RESERVE integer ALTERNATE AREAS format, you must specify the RESERVE clause with *integer* + *1* areas to get equivalent processing under COBOL/VSE. That is, the following specifications are equivalent:

DOS/VS COBOL	RESERVE 2 ALTERNATE AREAS
COBOL/VSE	RESERVE 3 AREAS

Under DOS/VS COBOL with LANGLVL(1), the interpretation of the RESERVE integer AREAS format differs from the interpretation of this format using COBOL/VSE.

With LANGLVL(1), using the RESERVE integer AREA or RESERVE integer AREAS format, you must specify the RESERVE clause with *integer* + *1* areas to get equivalent processing under COBOL/VSE.

Reserved Word List Changes

Differences exist between the reserved word list for COBOL/VSE and the reserved word list for DOS/VS COBOL. Appendix H, "COBOL Reserved Word Comparison" in topic APPENDIX1.8 contains a complete listing of reserved words for both products.

Reserved words for unsupported COBOL 85 Standard features are accepted as reserved words by COBOL/VSE, which flags them with a severe error message.

Additional CODASYL Reserved Words: The COBOL/VSE compiler flags the following additional CODASYL reserved words with an informational message and accepts them as user-defined words.

B-LESS	END-DISABLE
COMP-5	END-ENABLE
COMP-6	END-SEND
COMP-7	END-TRANSCEIVE
COMP-8	EXACT
COMP-9	INDEX-1

COMPUTATIONAL-5	INDEX-2
COMPUTATIONAL-6	INDEX-3
COMPUTATIONAL-7	INDEX-4
COMPUTATIONAL-8	INDEX-5
COMPUTATIONAL-9	INDEX-6
CONTAINED	INDEX-7
DISPLAY-2	INDEX-8
DISPLAY-3	INDEX-9
DISPLAY-4	PARAGRAPH
DISPLAY-5	SESSION-ID
DISPLAY-6	STANDARD-3
DISPLAY-7	STANDARD-4
DISPLAY-8	TRANSCEIVE
DISPLAY-9	

SEARCH Statement Changes

Under COBOL/VSE, the WHEN phrase data-name (the subject of the WHEN relation-condition) must be an ASCENDING/DESCENDING KEY data item in this table element, and identifier-2 (the object of the WHEN relation-condition) must not be an ASCENDING/DESCENDING key data item for this table element.

Under DOS/VS COBOL, these rules do not apply. The ASCENDING/DESCENDING KEY data item can be specified either as the subject or as the object of the WHEN relation-condition.

The following SEARCH example will execute under both COBOL/VSE and DOS/VS COBOL:

```
01  VAL  PIC X.
01  TABLE-01.
    05  TABLE-ENTRY
           OCCURS 100 TIMES
           ASCENDING KEY IS KEY-1
           INDEXED BY INDEX-NAME-1.
        10  FILLER PIC X.
        10  KEY-1 PIC X.
    SEARCH ALL TABLE-ENTRY
        AT END DISPLAY "ERROR"
      WHEN KEY-1 ( INDEX-NAME-1 ) = VAL
        DISPLAY "TABLE RECORDS OK".
```

DOS/VS COBOL accepts the following. COBOL/VSE does not:

```
  WHEN VAL = KEY-1 ( INDEX-NAME-1 )
    DISPLAY "TABLE RECORDS OK".
```

Segmentation Changes--PERFORM Statement in Independent Segments

In DOS/VS COBOL with LANGLVL(1), if a PERFORM statement in an

independent segment refers to a permanent segment, the independent segment is initialized upon each exit from the performed procedures.

In DOS/VS COBOL with LANGLVL(2), for a PERFORM statement in an independent segment that refers to a permanent segment, control is passed to the performed procedures only once for each execution of the PERFORM statement.

In COBOL/VSE, the compiler does not perform overlay; therefore, the rules given above do not apply.

If your program logic depends upon either of the DOS/VS COBOL implementations of these segmentation rules, you must rewrite the program.

SELECT OPTIONAL Clause Changes

In DOS/VS COBOL, the SELECT OPTIONAL clause of the File Control entry is treated as documentation.

In COBOL/VSE the SELECT OPTIONAL clause is required for sequentially organized labeled input files that are not necessarily present when the object programs runs. A file status code of 05 will be presented.

SORT Special Registers

The use of SORT special registers in COBOL/VSE is different than in DOS/VS COBOL, as shown below:

Figure 28. Sorting in DOS/VS COBOL and COBOL/VSE

To Specify	**In DOS/VS COBOL, Use**	**In COBOL/VSE, Use**
Sort completion code	Special register: SORT-RETURN	Special register: SORT-RETURN
Name of file with sort control statements	N/A	Special register: SORT-CONTROL
Modal length of records in a file with variable-length records	N/A	SMS control statement: SMS=*nnnnn* or Special register: SORT-MODE-SIZE
Number of sort records	N/A	Special register: SORT-FILE-SIZE Creates a SIZE keyword on the SORT CONTROL statement (ignored by DFSORT/VSE)
Amount of main storage to be used	SORT-OPTION clause keyword: STORAGE	OPTION control statement keyword: STORAGE or Special register: SORT-CORE-SIZE
Name of sort message file	SORT-OPTION clause keyword: ROUTE	OPTION control statement keyword: ROUTE or Special register: SORT-MESSAGE

See your sort manual for the meaning of keywords.

The SORT-CORE-SIZE, SORT-FILE-SIZE, SORT-MESSAGE, and SORT-MODE-SIZE special registers are supported under COBOL/VSE, and they will be used in the SORT interface when they have nondefault values. However, at run time, individual SORT special registers will be overridden by the corresponding parameters on control statements that are included in the SORT-CONTROL file, and a message will be issued. In addition, a compiler warning message (W-level) will be issued for each SORT special register that was set in the program.

In DOS/VS COBOL, the SORT-RETURN special register can contain codes for successful SORT completion (RC=0), OPEN or I/O errors concerning the USING or GIVING files (RC=2 through RC=12), and unsuccessful SORT completion (RC=16). In COBOL/VSE, the SORT-RETURN register only contains codes for successful (RC=0) and unsuccessful (RC=16) SORT completion.

SPECIAL-NAMES paragraph changes

In the SPECIAL-NAMES paragraph, there are migration considerations for the ALPHABET clause and the CURRENCY clause.

ALPHABET Clause Changes: In DOS/VS COBOL, and COBOL/VSE with CMPR2, the keyword ALPHABET is not allowed in the ALPHABET clause.

In COBOL/VSE with NOCMPR2, the ALPHABET keyword is required.

Also, in COBOL/VSE, *alphabet-name-1*, which specifies a character code set or collating sequence, may additionally be specified as:

EBCDIC	For EBCDIC
STANDARD-2	For the International Reference Version of the ISO 7-bit code (identical to STANDARD-1, except for the symbol "$")

CURRENCY-SIGN Changes: DOS/VS COBOL with LANGLVL(1) accepts the characters "/" and "=" in the CURRENCY-SIGN clause.

COBOL/VSE does not accept these characters as valid. In addition, COBOL/VSE (for programs that use DBCS data items) does not accept the character "G".

If these characters are present, you must remove them from the CURRENCY-SIGN clause.

Subscripts Out of Range Flagged at Compile Time

COBOL/VSE generates an E-level compile-time error message if a literal subscript or index value is coded that is greater than the allowed maximum, or less than one. This message is generated whether or not the SSRANGE option is specified.

DOS/VS COBOL does not generate an equivalent error message.

UNSTRING Statements--Subscript Evaluation Changes

In the UNSTRING statements for DOS/VS COBOL, any associated subscripting, indexing, or length calculation is evaluated immediately before the transfer of data into the receiving item for the DELIMITED BY, INTO, DELIMITER IN, and COUNT IN fields.

For these fields, in the COBOL/VSE UNSTRING statement, any associated subscripting, indexing, or length calculation is evaluated once--immediately before the examination of the delimiter sending fields.

For example:

```
01 ABC     PIC  X(30).
01 IND.
   02 IND-1 PIC 9.
01 TAB.
   02 TAB-1 PIC X OCCURS 10 TIMES.
01 ZZ      PIC  X(30).


   UNSTRING ABC DELIMITED BY TAB-1 (IND-1) INTO IND ZZ.
```

Under DOS/VS COBOL, subscript IND-1 is re-evaluated before the second receiver IND is filled.

Under COBOL/VSE, the subscript IND-1 is evaluated only once at the beginning of the execution of the UNSTRING statement.

Under DOS/VS COBOL with LANGLVL(1), when the DELIMITED BY ALL option of UNSTRING is specified, two or more contiguous occurrences of any delimiter are treated as if they were only one occurrence. As much of the first occurrence as will fit is moved into the current delimiter receiving field (if specified). Each additional occurrence is moved only if the complete occurrence will fit. For more information on the behavior of this option in DOS/VS COBOL, see *IBM VS COBOL for DOS/VSE*.

Under COBOL/VSE, one or more contiguous occurrences of any delimiters are treated as if they were only one occurrence, and this one occurrence is moved to the delimiter receiving field (if specified).

For example, if ID-SEND contains 123**45678**90AB:

```
UNSTRING ID-SEND DELIMITED BY ALL "*"
    INTO ID-R1 DELIMITER IN ID-D1 COUNT IN ID-C1
    INTO ID-R2 DELIMITER IN ID-D2 COUNT IN ID-C2
    INTO ID-R3 DELIMITER IN ID-D3 COUNT IN ID-C3
```

DOS/VS COBOL with LANGLVL(1), will produce this result:

```
ID-R1  123       1D-D1  **        ID-C1  3
ID-R2  45678     1D-D2  **        ID-C2  5
ID-R3  90AB      1D-D3            ID-C3  4
```

and COBOL/VSE will produce this result:

```
ID-R1  123       1D-D1  *         ID-C1  3
ID-R2  45678     1D-D2  *         ID-C2  5
```

```
ID-R3  90AB      1D-D3           ID-C3  4
```

UPSI Switches

DOS/VS COBOL allows references to UPSI switches and mnemonic names associated with UPSI. COBOL/VSE allows condition-names only. In DOS/VS COBOL, the condition-names can be qualified, but in COBOL/VSE, the condition-names must be unique.

For example, if a condition-name is defined in the SPECIAL-NAMES paragraph, the following will result:

DOS/VS COBOL:

```
SPECIAL-NAMES.
    UPSI-0 IS MNUPO
     .
     .
     .
PROCEDURE DIVISION
     .
     .
     .
    IF UPSI-0 = 1 ...
    IF MNUPO=0 ...
```

COBOL/VSE:

```
SPECIAL-NAMES.
    UPSI-0 IS MNUPO
     ON STATUS IS UPSI-0-ON
     OFF STATUS IS UPSI-0-OFF
     .
     .
     .
PROCEDURE DIVISION
     .
     .
     .
     IF UPSI-0-ON ...
     IF UPSI-0-OFF ...
```

UPSI switches may only be set using the LE/VSE UPSI run-time option. UPSI switches set using the VSE JCL UPSI statement will not be honored if a VS COBOL II or COBOL/VSE program is present in the application.

VS COBOL II and COBOL/VSE compiled programs are able to modify UPSI switch settings. However, the UPSI switch settings available to DOS/VS COBOL compiled programs will be those current at the time the program was invoked. Changes to UPSI switches made by VS COBOL II or COBOL/VSE programs called by a DOS/VS COBOL program will not be available to the DOS/VS COBOL program until it is re-invoked.

VALUE Clause Condition Names

For VALUE clause condition names, releases prior to Release 3.0 of DOS/VS COBOL allow the initialization of an alphanumeric field with a numeric value. For example:

```
01 FIELD-A PICTURE XX.
    88   LAST-YEAR   VALUE 87.
    88   THIS-YEAR   VALUE 88.
    88   NEXT-YEAR   VALUE 89.
```

COBOL/VSE does not accept this language extension. Therefore, to correct the above example, you must code alphanumeric values in the VALUE clauses, as in the following example:

```
01 FIELD-A PICTURE XX.
   88   LAST-YEAR   VALUE "87".
   88   THIS-YEAR   VALUE "88".
   88   NEXT-YEAR   VALUE "89".
```

WHEN-COMPILED Special Register

Both COBOL/VSE and DOS/VS COBOL support the use of the

WHEN-COMPILED special register. The rules for use of the special register are the same for both compilers. However, the format of the data differs.

In DOS/VS COBOL, the format is:

MM/DD/YYhh.mm.ss

(MONTH/DAY/YEARhour.minute.second)

or, (dependent on the VSE system date format):

DD/MM/YYhh.mm.ss

(DAY/MONTH/YEARhour.minute.second)

In COBOL/VSE, the format is:

MM/DD/YYhh.mm.ss

(MONTH/DAY/YEARhour.minute.second)

WRITE AFTER POSITIONING Statement

DOS/VS COBOL supports the WRITE statement with the AFTER POSITIONING option. COBOL/VSE does not.

If *n* is a literal, this is changed to WRITE...AFTER ADVANCING *n* LINES. If *n* is an identifier, SPECIAL-NAMES are generated and a section is added at the end of the program.

DOS/VS COBOL	**COBOL/VSE**
AFTER POSITIONING 0	AFTER ADVANCING PAGE
AFTER POSITIONING 1	AFTER ADVANCING 1
AFTER POSITIONING 2	AFTER ADVANCING 2
AFTER POSITIONING 3	AFTER ADVANCING 3

```
WRITE OUTPUT-REC AFTER POSITIONING SKIP-CC.
```

DOS/VS COBOL	SKIP-CC	**COBOL/VSE**
AFTER POSITIONING SKIP-CC	1	AFTER ADVANCING PAGE
AFTER POSITIONING SKIP-CC	' '	AFTER ADVANCING 1
AFTER POSITIONING SKIP-CC	0	AFTER ADVANCING 2
AFTER POSITIONING SKIP-CC	-	AFTER ADVANCING 3

Note: When compiling the converted program with COBOL/VSE, use the NOADV option. If POSITIONING and ADVANCING are used in the old program, you should review the ADV option.

WRITE AT END-OF-PAGE Statement Changes

DOS/VS COBOL allows the END-OF-PAGE option of the WRITE statement to be specified for unblocked, single buffered files even when the FD entry for the file does not contain the LINAGE clause.

COBOL/VSE only allows the END-OF-PAGE option to be specified when the FD entry for the file contains the LINAGE clause.

Therefore, you must recode your program to remove the dependency on the END-OF-PAGE condition being detected for files that do not contain the LINAGE clause, or add the LINAGE clause to the necessary FD entries.

4.2 Chapter 8. Compiling Your Migrated DOS/VS COBOL Programs

This chapter describes the differences in compile-time environments between DOS/VS COBOL and COBOL/VSE and gives advice about actions to take to make conversion as efficient as possible.

This chapter contains information on the following topics:

- Using the COBOL/VSE compiler options
- Other compile-time conversion considerations

Subtopics:

4.2.1 Recommended COBOL/VSE Compiler Options

The following paragraphs describe the COBOL/VSE compiler options that have special relevance to migrated programs.

Figure 29. Recommended COBOL/VSE Compiler Options

COBOL/VSE Compiler Option	Comments
BUFSIZE	In COBOL/VSE, this option specifies the amount of buffer storage reserved for each compiler work file. The default is 4096. In DOS/VS COBOL, the comparable BUF option has a

	default value of 512. (The amount of virtual storage allocated for buffers is included in the amount of storage available to the compiler through the SIZE option.)
DATA(24)	Use DATA(24) for applications that contain both DOS/VS COBOL and COBOL/VSE programs in the same run unit.
NUMPROC(MIG)	NUMPROC(MIG) processes numeric signs in a way similar to DOS/VS COBOL.
OUTDD(filename)	Use this option to replace the default filename (SYSLST) for SYSLST output that goes to the system logical output unit. If the filename is the same as the LE/VSE MSGFILE filename, the output will be routed to the filename designated for the MSGFILE.
RMODE(24)	Use RMODE(24) for COBOL/VSE NORENT programs that pass data to programs running in AMODE(24).
TRUNC	This compiler option controls the way arithmetic fields are shortened into binary receiving fields during MOVE and arithmetic operations. The DOS/VS COBOL TRUNC option corresponds to the COBOL/VSE TRUNC(STD) option--it causes the sending data to be shortened to the number of digits specified in the PICTURE clause of the binary receiver. The DOS/VS COBOL NOTRUNC option has no exact corresponding option under COBOL/VSE. The TRUNC(OPT) option gives similar results as the DOS/VS COBOL NOTRUNC option, except in a limited number of cases where a W-level error message is generated. Assuming the binary data conforms to its PICTURE specification, the TRUNC(OPT) option offers the best performance. The TRUNC(BIN) option is recommended for selected applications that require guaranteed nontruncation of binary data. This is particularly true if there is a possibility that data being moved into binary data items can have a value larger than that defined by the PICTURE clause for the binary data item.

NUMCLS Option

The NUMCLS option can be specified only at installation time. This compiler option, together with the NUMPROC option, affects execution of the numeric class test. Figure 30 and Figure 31 show the sign representations recognized for DISPLAY, COMP-3, and separately signed data.

Figure 30 shows sign representations recognized when NUMCLS(PRIM) is in effect. Figure 31 shows sign representations recognized when NUMCLS(ALT) is in effect.

Figure 30. Sign Representation with NUMCLS(PRIM)

	NUMPROC(NOPFD)	**NUMPROC(PFD)**	**NUMPROC(MIG)**
Signed	C, D, F	C(1), D	C, D, F
Unsigned	F	F	F

Separate Sign	+, -	+(2), -,	+, -

Figure 31. Sign Representation with NUMCLS(ALT)

	NUMPROC(NOPFD)	NUMPROC(PFD)	NUMPROC(MIG)
Signed	A to F	C(1), D	A to F
Unsigned	F	F	F
Separate Sign	+, -	+(2), -	+, -

Note:

(1) If the value is zero, the sign must be "C".

(2) If the value is zero, the sign must be "+".

Subtopics:

4.2.1.1 Process COBOL/VSE Compiler Options

When the COBOL/VSE compiler translates your source program into machine code, you can control compilation results using the compiler options that COBOL/VSE provides. The compiler options let you:

- Control object code generation
- Control compiler usage of virtual storage
- Control listings, maps, and diagnostics
- Provide run-time debugging information
- Set the delimiter for literals
- Select a customized reserved word list
- Process COPY or BASIS statements

Your site can use several different methods to control COBOL/VSE compilations. The methods are listed below, in their order of precedence:

1. Options "fixed" by your site at installation through the IGYCINS macro (the application programmer cannot replace these options)

2. Options specified on CBL statements

3. Options specified by the PARM parameter of the JCL EXEC statement

4. Options specified on the JCL OPTION statement

5. Default options established during installation (options not "fixed" by your site). Unless you replace them by specifying other options, your compilation uses these default options

If you specify a COBOL/VSE compiler option that conflicts with another, COBOL/VSE generates a diagnostic message. The message describes the problem and also describes how the compiler resolved the conflict.

In general, the COBOL/VSE compiler (like the DOS/VS COBOL compiler) uses the last encountered version of an option within each of the five methods.

With two exceptions, you can specify all the compiler options on a PROCESS(CBL) statement at the beginning of your program. The DECK option can only be specified using the JCL OPTION DECK. The OBJECT option can only be specified using the JCL OPTIONs LINK and CATAL.

PARM and PROCESS(CBL) statement options are processed from left to right. For compiler option default values, see the *COBOL/VSE Programming Guide*. For information on customizing the compiler options for your site, see *COBOL/VSE Installation and Customization Guide*.

4.2.1.2 DOS/VS COBOL Compiler Options Not Supported in COBOL/VSE

Figure 32 shows the DOS/VS COBOL compiler options that are not supported by COBOL/VSE. COBOL/VSE equivalents (if any) are described.

For a complete list of the COBOL/VSE compiler options, see Appendix F, "Compiler Option Comparison" in topic APPENDIX1.6.

Figure 32. DOS/VS COBOL Compiler Options Not Supported by COBOL/VSE

DOS/VS COBOL Option	COBOL/VSE Equivalent (If Any)
BUF=nnnn	Equivalent to COBOL/VSE compiler option BUFSIZE(n)
CATALR/NOCATALR	Equivalent to COBOL/VSE compiler option NAME/NONAME
CLIST/NOCLIST	Equivalent to COBOL/VSE compiler option OFFSET/NOOFFSET
COUNT/NOCOUNT	No equivalent option in COBOL/VSE.
FLAGE/FLAGW	Equivalent to COBOL/VSE compiler options FLAG(E) and FLAG(W)
FLOW/NOFLOW	No equivalent option in COBOL/VSE.
LANGLVL(1/2)	LANGLVL option does not exist in COBOL/VSE. COBOL/VSE supports only the COBOL 85 Standard and the COBOL 74 Standard as implemented by VS COBOL II Release 2.
LVL=A|B|C|D/NOLVL	ANSI COBOL 74 FIPS is not supported in

	COBOL/VSE.
MIGR/NOMIGR	MIGR is not required for COBOL/VSE
PMAP=h	Obsolete option. No COBOL/VSE equivalent.
SPACEn	Equivalent to COBOL/VSE compiler option SPACE(n).
STATE/NOSTATE	Function is available using COBOL/VSE compiler option TEST.
SUPMAP/NOSUPMAP	Equivalent to COBOL/VSE compiler option NOCOMPILE/COMPILE.
STXIT/NOSTXIT	Function not required in COBOL/VSE
SXREF/NOSXREF	COBOL/VSE XREF option provides sorted SXREF output.
SYMDMP/NOSYMDMP	Abend dumps and dynamic dumps are available through LE/VSE services. Symbolic dumps are available through COBOL/VSE, using the TEST compiler option.
SYNTAX/CSYNTAX/NOSYNTAX	Equivalent to COBOL/VSE compiler option NOCOMPILE/NOCOMPILE(S)
VERB/NOVERB	Function is available in COBOL/VSE compiler option LIST/NOLIST.
VERBREF/NOVERBREF	Equivalent to COBOL/VSE compiler option VBREF/NOVBREF
VERBSUM/NOVERBSUM	Function available in COBOL/VSE compiler option VBREF
DECK/NODECK (LISTER)	The LISTER feature is not supported in COBOL/VSE.
COPYPCH/NOCOPYPCH	The LISTER feature is not supported in COBOL/VSE.
LSTONLY/LSTCOMP/NOLST	The LISTER feature is not supported in COBOL/VSE.
PROC=1|2col	The LISTER feature is not supported in COBOL/VSE.

4.2.2 Other Compile-Time Conversion Considerations

The following compile-time conversion considerations apply:

- Paragraph names not allowed as parameters
- Prolog format changes

Subtopics:

- 4.2.2.1 Paragraph Names Not Allowed as Parameters
- 4.2.2.2 Prolog Format Changes

4.2.2.1 Paragraph Names Not Allowed as Parameters

DOS/VS COBOL allows paragraph names and nonsequential file names to be passed to an assembler module. COBOL/VSE does not support these functions.

4.2.2.2 Prolog Format Changes

Object modules generated by COBOL/VSE are LE/VSE conforming, and therefore have a different prolog format than in DOS/VS COBOL. You will need to update existing applications that scan for date and time to the new format.

You can compile your programs with the COBOL/VSE LIST compiler option to generate a listing that you can use to compare the DOS/VS COBOL format with the COBOL/VSE format.

4.3 Chapter 9. Modifying Your VS COBOL II Source Programs

If your VS COBOL II source programs were compiled with the NOCMPR2 compiler option, they will most likely compile under COBOL/VSE without change. However, you might need to make modifications based on:

- COBOL 85 Standard interpretation changes
- Reserved Words changes

If your VS COBOL II source programs were compiled with the CMPR2 compiler option, it is recommended that you convert them to NOCMPR2 programs in order to take advantage of the language enhancements provided by the COBOL 85 Standard, as well as to position for future Standard enhancements. For information on language differences between CMPR2 and NOCMPR2 (COBOL 85 Standard) see the *VS COBOL II Migration Guide for VSE*.

Subtopics:

4.3.1 COBOL 85 Standard Interpretation Changes

Some language differences exist between VS COBOL II Release 3.2 (NOCMPR2), and COBOL/VSE. These changes are the result of responses from COBOL Standard Interpretation Requests that required an implementation different from that used in VS COBOL II. The following language elements are affected:

- REPLACE and comment lines
- Precedence of USE procedures
- Reference modification of a variable-length group

Note: VS COBOL II Release 4 includes these Standard interpretations. For other releases of VS COBOL II, the CCCA/VSE Program Offering (Release 1) can, in some cases, convert these statements automatically. When automatic conversion is not possible, CCCA will flag statements that require manual conversion.

Subtopics:

4.3.1.1 REPLACE and Comment Lines

This item affects the treatment of blank lines and comment lines that appear in text that matches pseudo-text-1 of REPLACE statements.

Blank lines, which are interspersed in the matched text, will not appear in the output of the REPLACE statement. This could affect the semantics of the resulting program since the line numbers could be different. (For example, if the program uses the USE FOR DEBUGGING declarative, the contents of DEBUG-ITEM might be different). If the COBOL/VSE generated program differs from the equivalent VS COBOL II program, the following message will be issued:

IGYLI0193-I Matched pseudo-text-1 contained blank or comment lines. Execution results may differ from VS COBOL II.

4.3.1.2 Precedence of USE Procedures

This difference affects the precedence of USE procedures relating to contained programs.

In the VS COBOL II implementation, a file-specific USE procedure always takes precedence over a mode-specific USE procedure, even if an applicable mode-specific USE procedure exists in the current program, or if a mode-specific USE procedure with the GLOBAL attribute in an outer program is "nearer" than the file-specific procedure.

In COBOL/VSE, USE procedure precedence is based on a program-by-program level, from the current program to its containing program, and so on, to the outermost program.

If the COBOL/VSE generated program selects a different USE procedure than would have been used by the VS COBOL II program, the following message will be issued:

IGYSC2300-I A mode-specific declarative may be selected for file "file-name" in program "program-name." Execution

results may differ from VS COBOL II.

4.3.1.3 Reference Modification of a Variable-Length Group

This difference relates to the value used for the length of a variable-length group receiver that contains its own OCCURS DEPENDING ON (ODO) object and is reference-modified with a reference modifier that does **not** specify a length.

VS COBOL II Release 3.2 used the actual length of the group as defined by the current value of the ODO object. COBOL/VSE uses the maximum length of the group, as defined by integer-2 of the OCCURS integer-1 to integer-2 clause. (The maximum number of occurrences.)

If the program contains a reference-modified, variable-length group receiver that contains its own ODO object and whose reference modifier does not have a length specified, the following message is issued:

IGYPS2298-I The reference to variable-length group "data name" will be evaluated using the maximum length of the group. Execution results may differ from VS COBOL II.

The following code would give **different** results with COBOL/VSE than with VS COBOL II:

```
01 A.
   02 A-OBJ   PIC 99.
   02 A-TAB   PIC X(4) OCCURS 10 TIMES DEPENDING ON A-OBJ.
    .
    .
    .
MOVE ALL SPACES TO A(1:).
```

The following code would give the **same** results with COBOL/VSE as with VS COBOL II:

```
MOVE ALL SPACES TO A(1:LENGTH OF A).
```

Note: VS COBOL II Release 4 includes these standard interpretations. For other releases of VS COBOL II, both the COBOL and CICS Command Level Conversion Aid for VSE (CCCA) (Release 1 or later) and COBOL/VSE flag the REPLACE and USE procedures. CCCA automatically converts the reference modification group receiver that contains its own OCCURS DEPENDING ON object.

4.3.2 Reserved Word Changes

Two new reserved words have been added to COBOL/VSE: PROCEDURE-POINTER and FUNCTION. If either of these new Reserved Words are used in your current programs, you must remove them.

4.4 Chapter 10. Compiling Your Migrated VS COBOL II Programs

This chapter describes the differences that exist between the COBOL/VSE and VS COBOL II compile-time environment. This chapter contains information on the following topics:

- Compiler option considerations
- Prolog format changes

Each section describes the differences, if any, between the VS COBOL II and the COBOL/VSE implementations.

Subtopics:

4.4.1 VS COBOL II Compiler Option Considerations

The VS COBOL II and the COBOL/VSE compile-time environments are very similar. If you use the same compiler options that are specified in your current VS COBOL II applications, some internal changes might take effect, but basically the behavior is unchanged.

If you do change compiler option settings in your VS COBOL II applications, make sure you understand the possible effects on your applications. For information on migrating from CMPR2 to NOCMPR2, see the *VS COBOL II Migration Guide for VSE*. For information on other compiler options, see the *COBOL/VSE Programming Guide*.

Changes to VS COBOL II compiler options are described below. Note, the CCCA/VSE Program Offering (Release 1 or later) can convert these compiler options automatically.

RES

COBOL/VSE does not support the RES compiler option. Now, the object module is always created such that library subroutines are loaded dynamically at run time instead of being link-edited with the COBOL program. This is equivalent to RES behavior in VS COBOL II.

If RES is encountered by the COBOL/VSE compiler, an informational message is returned by COBOL/VSE.

FDUMP

COBOL/VSE does not support the FDUMP compiler option. For existing applications, FDUMP is mapped to the COBOL/VSE TEST(NONE,SYM) compiler option, which can provide equivalent function and more.

LE/VSE generates a better formatted dump than VS COBOL II, regardless of the FDUMP option. But, the presence of FDUMP

enables LE/VSE to include the symbolic dump of information about data items in the formatted dump.

For information on how to obtain the LE/VSE-formatted dump at abnormal termination, see *LE/VSE Debugging Guide and Run-Time Messages*.

If NOFDUMP has been specified, an informational message 4003 will be issued by COBOL/VSE because NOFDUMP is no longer supported.

The STAE run-time option, or its equivalent (LE/VSE TRAP(ON)), is no longer needed to generate a formatted dump.

TEST

In COBOL/VSE, the syntax of this option has changed. The TEST option now has two suboptions; instead of specifying TEST, you now can specify a hook location and symbol-table location.

TEST without any suboptions gives you TEST(ALL,SYM). For more information on the TEST option, see *COBOL/VSE Programming Guide*.

4.4.2 Prolog Format Changes

Object modules generated by COBOL/VSE are LE/VSE conforming, and thus have a different prolog format than in VS COBOL II. Existing applications scanning for date/time and user level information need to be updated to the new format.

You can compile your programs with the COBOL/VSE LIST compiler option to generate a listing that you can use to compare the VS COBOL II format with the COBOL/VSE format.

4.5 Chapter 11. Migrating Subsystem Source Programs

This chapter explains the differences in source language considerations for programs running under the following subsystems:

- CICS
- DL/I
- SQL/DS

This chapter first gives you a list of the recommended compiler options for running under CICS. Second, it gives details on source language considerations.

Subsystem run-time considerations are included in: Chapter 5, "Moving from the DOS/VS COBOL Run Time to LE/VSE" in topic 3.2 and Chapter 6, "Moving from the VS COBOL II Run Time to LE/VSE" in topic 3.3.

Subtopics:

4.5.1 CICS Conversion Considerations

The CICS environment supports the use of COBOL/VSE and LE/VSE, beginning with CICS Version 2 Release 3 or later. You can compile and run CICS command-level and macro-level application programs.

This section describes the action you need to take for applications which use source language (either CICS source or DOS/VS COBOL source), that involve the following function:

- COBOL/VSE language elements not supported under CICS
- Suspending/restoring CICS commands
- Optional coding changes
- DOS/VS COBOL base addressability

Subtopics:

4.5.1.1 Recommended Compiler Options for CICS

Figure 33 lists the recommended compiler options for COBOL/VSE programs running under CICS.

Figure 33. Recommended Compiler Options for CICS

On CICS	Comments
DATA(24)/(31)	Use DATA(24) in the following cases: ° When you link-edit any batch program that is going to use CICS shared database (DFHDRP), since the parameters passed to DFHDRP must be below the 16-megabyte line.
NODYNAM	Use NODYNAM for COBOL/VSE programs translated by the CICS translator.
RENT	Required for CICS programs. RENT causes COBOL/VSE to produce reentrant code and allows you to place the COBOL modules in the SVA (Shared Virtual Area) to be shared among multiple address spaces under CICS.
TRUNC(OPT) TRUNC(BIN)	Use TRUNC(OPT) for CICS programs containing EXEC CICS commands, if the program's use of binary data items conforms to the PICTURE and USAGE clause for those data items.

	Use TRUNC(BIN) if your program's use of binary data items does not conform to the PICTURE and USAGE clause for those data items, for example, if you have a data item defined as PIC S9(8) BINARY that might receive a value greater than eight digits. **Note:** Using TRUNC(BIN) might adversely affect the performance of the application. For more information, see the *COBOL/VSE Programming Guide*.

4.5.1.2 COBOL/VSE Language Elements Not Supported under CICS

The COBOL/VSE TIME and DATE conceptual data items are not supported under CICS.

4.5.1.3 CICS Language Elements that Suspend/Restore CICS Commands

The CICS PUSH HANDLE and POP HANDLE commands suspend/restore the current effects of the following CICS commands:

- IGNORE CONDITION
- HANDLE ABEND
- HANDLE AID
- HANDLE CONDITION

VS COBOL II issues the PUSH HANDLE and POP HANDLE commands automatically on calls to COBOL subroutines. To receive this same behavior in LE/VSE, you must specify the LE/VSE CBLPSHPOP(ON) run-time option.

CBLPSHPOP(OFF) prevents the issuing of the PUSH HANDLE and POP HANDLE commands and must be used with care to avoid compatibility problems when upgrading. If your application calls COBOL (VS COBOL II or COBOL/VSE) subroutines under CICS, performance is better with CBLPSHPOP(OFF) than with CBLPSHPOP(ON). (You can set the CBLPSHPOP run-time option on a transaction-by-transaction basis by using CEEUOPT.)

If you do not issue a CICS PUSH HANDLE command before a call to a COBOL subroutine, the subroutine will inherit any IGNORE CONDITION or HANDLE command from the caller. If the subroutine then issues an IGNORE CONDITION or HANDLE command, the caller will inherit their effects upon return.

To avoid compatibility problems, specify CBLPSHPOP(ON) if either of the following conditions are present:

1. The caller contains either of these CICS commands:
   - CICS IGNORE CONDITION
   - CICS HANDLE ABEND PROGRAM(program)
2. The COBOL subroutine contains any of the "PUSHable" CICS commands:
   - CICS IGNORE CONDITION
   - CICS HANDLE ABEND

- CICS HANDLE AID
- CICS HANDLE CONDITION

Note that CICS HANDLE...(label) commands in the caller will not cause compatibility problems with CBLPSHPOP(OFF) as long as your program does not contain any of the statements listed above.

For example, if the caller contains an EXEC CICS HANDLE ABEND LABEL(LBL1) and the subroutine incurs an abend, LE/VSE will prevent a branch to LBL1 across program boundaries and the application will abend (the same behavior as with CBLPSHPOP(ON)).

EXAMPLE 1: If CBLPSHPOP(ON) is specified, no HANDLE ABEND command is in effect for PGM2, which will lead to an ASRA abend caused by the attempt to divide by zero.

If CBLPSHPOP(OFF) is specified, the exception in PGM2 will cause CICS to attempt to branch to LBL1 in PGM1, which LE/VSE will prevent. The program will terminate with an ASRA abend.

```
       PROGRAM-ID. PGM1
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       PROCEDURE DIVISION.
           EXEC CICS HANDLE ABEND LABEL(LBL1) END-EXEC.
           CALL "PGM2" USING DFHEIBLK DFHCOMMAREA
           EXEC CICS RETURN END-EXEC.
       LBL1.
           EXEC CICS RETURN END-EXEC.
```

```
       PROGRAM-ID. PGM2
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  DATA1    PIC 9 VALUE 0.
       01  DATA2    PIC 9 VALUE 1.
       PROCEDURE DIVISION.
           COMPUTE DATA1 = DATA1 / DATA2
           GOBACK.
```

EXAMPLE 2: With CBLPSHPOP(ON), PGM1's HANDLE ABEND label is saved and restored across the call to PGM2, and the divide by zero exception is handled with a branch to LBL1. The program terminates normally.

With CBLPSHPOP(OFF), PGM2's HANDLE ABEND command is in effect on the return to PGM1. The exception caused by the divide by zero in PGM1 causes CICS to attempt to branch to LBL1A in PGM2, which LE/VSE prevents. The program terminates with an ASRA abend.

```
       PROGRAM-ID. PGM1
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  DATA1    PIC 9 VALUE 0.
       01  DATA2    PIC 9 VALUE 1.
       PROCEDURE DIVISION.
```

```
            EXEC CICS HANDLE ABEND LABEL(LBL1) END-EXEC.
            CALL "PGM2" USING DFHEIBLK DFHCOMMAREA
            COMPUTE DATA1 = DATA1 / DATA2
            EXEC CICS RETURN END-EXEC.
        LBL1.
            EXEC CICS RETURN END-EXEC.



        PROGRAM-ID. PGM2
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        PROCEDURE DIVISION.
            EXEC CICS HANDLE ABEND LABEL(LBL1A) END-EXEC.
            GOBACK.
        LBL1A.
            GOBACK.
```

4.5.1.4 Optional Coding Changes

With COBOL/VSE, some restrictions have been removed for language elements. In addition, COBOL/VSE provides some enhancements that you might want to use when upgrading your code to COBOL/VSE.

The following language elements are affected:

- Static and Dynamic Calls
- LENGTH OF Special Register
- Language Elements that require Dynamic Storage
- SERVICE RELOAD Statement

Subtopics:

4.5.1.4.1 Using Dynamic and Static Calls

The use of static and dynamic calls is enhanced with COBOL/VSE, although LE/VSE does not support static or dynamic calls between COBOL/VSE programs and DOS/VS COBOL programs. (Continue to access DOS/VS COBOL subroutines by EXEC CICS LINK.)

Using the Static CALL Statement: With DOS/VS COBOL, if multiple COBOL programs are separately compiled and then link-edited together, only the first program can contain CICS requests. With COBOL/VSE, this restriction is removed, giving you greater flexibility in application program design.

COBOL/VSE fully supports static calls from COBOL/VSE application programs running under CICS, and calls to other COBOL/VSE or assembler language programs that can contain CICS requests.

If your COBOL/VSE program was processed by the CICS translator and issues a static call, the calling program must pass the called COBOL/VSE program the CICS EXEC interface block (DFHEIBLK) and the communication area (DFHCOMMAREA) as the first two parameters of the CALL statement. If the COBOL/VSE program was not processed by the CICS translator, you only need to pass DFHEIBLK and DFHCOMMAREA if they are explicitly coded in the called program.

The CICS command language translator automatically inserts these parameters as the first two parameters on the corresponding PROCEDURE DIVISION USING statement in the called program.

Using the Dynamic CALL Statement: With COBOL/VSE you can make dynamic calls to subprograms, and the dynamically called subprograms can contain CICS commands.

For COBOL/VSE programs running under CICS, the ON EXCEPTION/OVERFLOW clause of the CALL statement is enabled for programs compiled with the NOCMPR2 compiler option.

4.5.1.4.2 Using the LENGTH OF Special Register

Through the LENGTH OF special register, you no longer need to pass explicit length arguments on many of the CICS commands where length is required. You can use the LENGTH OF special register in COBOL statements as if it were an explicitly defined numeric data item. This lets you obtain information about the number of characters a data item occupies in the program.

4.5.1.4.3 Using Language Elements That Require Dynamic Storage

In DOS/VS COBOL CICS programs, you cannot use STRING, UNSTRING, and INSPECT statements, or the USE FOR DEBUGGING declarative. With COBOL/VSE and LE/VSE, you can.

4.5.1.4.4 SERVICE RELOAD Statement Changes

In DOS/VS COBOL, for programs to be executed under CICS, the SERVICE RELOAD statement is required to ensure addressability of items defined in the linkage section.

In COBOL/VSE the SERVICE RELOAD statement is not required, and is treated as a comment.

4.5.1.5 Base Addressability Considerations

Using DOS/VS COBOL, you must maintain addressability to storage areas not contained within the Working-Storage section of the COBOL/CICS program. To satisfy program requests, a DOS/VS COBOL program must keep track of storage area addresses allocated by CICS. This requires the manipulation of the BLL cells within the application program.

With COBOL/VSE and the associated support within CICS, this is no longer necessary. Therefore, when you migrate from DOS/VS COBOL to COBOL/VSE, you must convert such programs. (If the COBOL/CICS program does not manipulate BLL cell addressing, conversion is not necessary.)

Note: You can do this automatically using CCCA, as explained in Appendix B, "Conversion Aids for Source Programs" in topic APPENDIX1.2.

Converting DOS/VS COBOL CICS Programs to COBOL/VSE CICS Programs: The following steps summarize the conversion of a DOS/VS COBOL CICS program to a COBOL/VSE CICS program. For more information, see *CICS/VSE Application Programming Guide*.

1. Remove all SERVICE RELOAD statements. The COBOL/VSE compiler treats these statements as comments. (Although desirable, removal is not absolutely necessary.)

2. Remove all operations dealing with addressing structures in the linkage section greater than 4K bytes in size. A typical statement is:

```
ADD +4096 D-PTR1 GIVING D-PTR2.
```

3. Remove all program code that assists in addressing COMMAREAs greater than 4K in size.

4. Remove redundant assignments and labels that DOS/VS COBOL uses to ensure that CICS programs are correctly optimized. (This is good programming practice, but it is not essential.)

   Redundant assignments and labels include:

   - Artificial paragraph names, ones that use BLL cells to address chained storage areas.

   - Artificial assignments from the object of an OCCURS ... DEPENDING ON clause to itself.

5. Change every SET(P) option in the CICS commands to SET(ADDRESS OF L), where "L" is the linkage section structure that corresponds to the "P" BLL cell.

6. Specify REDEFINES clauses in the linkage section, if multiple record formats are defined through the SET option.

7. Review programs that use Basic Mapping Support (BMS) data structures in their linkage section (check for maps that are not defined as STORAGE=AUTO). Move any maps not defined as STORAGE=AUTO to the Working-Storage section and remove any associated EXEC CICS GETMAIN commands.

DL/I Call Interface: After you have migrated to COBOL/VSE, you might want to continue using CALL CBLTDLI. To do so, review your programs and make the following changes:

1. Remove BLL cells for addressing the User Interface Block (UIB) and Program Communication Blocks (PCBs).

2. In the linkage section, retain the DLIUIB declaration and at least one PCB declaration.

3. Change the PCB call to specify the UIB directly, as follows:

```
CALL "CBLTDLI" USING PCB-CALL,
                     PSB-NAME,
                     ADDRESS OF DLIUIB
```

4. Obtain the address of the required PCB from the address list in the UIB.

Subtopics:

4.5.1.5.1 Example 1: Receiving a Communications Area

In this example, a COBOL program defines a record area within the linkage section. This general technique is used in a number of COBOL/CICS programs that define structures outside of the Working-Storage section.

During conversion, do the following:

1. Remove the address list definition defining the BLL cells; under COBOL/VSE, BLL cells are no longer explicitly defined in the linkage section.

2. In the SET option of the CICS command, use the ADDRESS OF special register when referring to the storage area, instead of specifying the BLL cell name.

DOS/VS COBOL **COBOL/VSE**

```
LINKAGE SECTION.                                  LINKAGE SECTION.
 01  PARAMETER-LIST.                              01  AREA1.
    05  PARM-FILLER          PIC S9(8) COMP.        05  AREA1-DATA     PIC X(100).
    05  PARM-AREA1-POINTER   PIC S9(8) COMP.      01  AREA2.
    05  PARM-AREA2-POINTER   PIC S9(8) COMP.         05  AREA2-DATA     PIC X(100).
 01  AREA1.                                         .
    05  AREA1-DATA           PIC X(100).           .
 01  AREA2.                                        PROCEDURE DIVISION.
    05  AREA2-DATA           PIC X(100).           .
          .                                       .
          .                                       EXEC CICS READ DATASET("INFILE")
  PROCEDURE DIVISION.                                 RIDFLD(INFILE-KEY)
          .                                           SET(ADDRESS OF AREA1)
          .                                           LENGTH(RECORD-LEN).
    EXEC CICS READ DATASET("INFILE")
       RIDFLD(INFILE-KEY)
       SET(PARM-AREA1-POINTER)
       LENGTH(RECORD-LEN)
    SERVICE RELOAD PARM-AREA1-POINTER.
```

4.5.1.5.2 Example 2: Processing Storage Areas Exceeding 4K

If a linkage section area is greater than 4096 bytes in length, you must add statements that provide addressability to the entire storage area.

The following example shows the coding for both a DOS/VS COBOL program and a COBOL/VSE program.

During conversion, do the following:

1. Remove the following statements used to maintain addressability:

```
ADD +4096 TO RECORD-POINTER ....
SERVICE RELOAD .....
```

2. Change the SET option of the CICS command from an intermediate BLL cell to the ADDRESS OF special register for the record.

DOS/VS COBOL

```
LINKAGE SECTION.
 01 PARMLIST.
    .
    .
    05  RECORD-POINTERA  PIC S9(8) COMP.
    05  RECORD-POINTERB  PIC S9(8) COMP.
    .
```

COBOL/VSE

```
LINKAGE SECTION.
 01 FILE-RECORD.
    05  REC-DATA1       PIC X(2500).
    05  REC-DATA2       PIC X(2500).
        .
        .
PROCEDURE DIVISION.
```

```
    .
 01 FILE-RECORD.
    05  REC-AREA1         PIC X(2500).
    05  REC-AREA2         PIC X(2500).
    .
    .
PROCEDURE DIVISION.
    .
    .
    EXEC CICS READ DATASET("INFILE")
       RIDFLD(INFILE-KEY)
       SET(RECORD-POINTERA)
       LENGTH(RECORD-LEN)
    END-EXEC
    SERVICE RELOAD RECORD-POINTERA
    ADD +4096 TO RECORD-POINTERA GIVING RECORD-POINTERB
    SERVICE RELOAD RECORD-POINTERB.
```

```
    .
    .
EXEC CICS READ DATASET("INFILE")
     RIDFLD(INFILE-KEY)
     SET(ADDRESS OF FILE-RECORD)
     LENGTH(RECORD-LEN)
END-EXEC
```

4.5.1.5.3 Example 3: Accessing Chained Storage Areas

In a DOS/VS COBOL CICS program, you can chain storage areas together by defining a storage area in the linkage section that contains a pointer to another storage area. To do this, you must copy the address of the next chained area from its location in one area to its associated BLL cell. In addition, you must code a paragraph name following any statement that changes the contents of the BLL cell used to address the chained areas.

During conversion, do the following:

1. Change the code that moves the next address from within a storage area to the associated BLL cell. Perform the identical function in COBOL/VSE by using the ADDRESS OF special register associated with the current and next storage areas.

2. If you like, remove the dummy paragraph names that follow references that change the contents of BLL cells. (This is good programming practice, but it is not essential.)

DOS/VS COBOL

```
WORKING-STORAGE SECTION.
01 WSDATA-HOLD PIC X(100).
    .
    .
LINKAGE SECTION.
01 PARAMETER-LIST.
    .
    .
    05  CHAINED-POINTER    PIC S9(8) COMP.
    .
    .
01 CHAINED-STORAGE.
    05  CHS-NEXT-AREA      PIC S9(8) COMP.
    05  CHS-DATA           PIC X(100).
    .
```

COBOL/VSE

```
WORKING-STORAGE SECTION.
01 WSDATA-HOLD PIC X(100).
      .
      .
 LINKAGE SECTION.
      .
      .
 01 CHAINED-STORAGE.
      05  CHS-NEXT-AREA       USAGE IS POINTER.
      05  CHS-DATA            PIC X(100).
      .
      .
 PROCEDURE DIVISION.
      .
      .
```

```
    .                                                  SET ADDRESS OF CHAINED-STORAGE TO CHS-NEXT-AREA
PROCEDURE DIVISION.                                    MOVE CHS-DATA TO WS-DATA-HOLD
    .
    .
    MOVE CHS-NEXT-AREA TO CHAINED-POINTER.
 ANY-PARAGRAPH-NAME.
    MOVE CHS-DATA TO WSDATA-HOLD.
    .
    .
```

4.5.1.5.4 Example 4: Using the OCCURS DEPENDING ON Clause

In DOS/VS COBOL, if the linkage section contains the object of the OCCURS DEPENDING ON clause, you must ensure that the correct length of the group is used when the number of occurrences of the OCCURS clause's subject changes.

During conversion, in your COBOL/VSE program, remove any code that resets the contents of the object of the OCCURS DEPENDING ON clause. (These references are no longer necessary.)

DOS/VS COBOL

```
LINKAGE SECTION.
  01 PARMLIST.
     05  FILLER                PIC S9(8).
     05  RECORD-POINTER        PIC S9(8).
     .
     .
  01  VAR-RECORD.
     05  REC-OTHER-DATA        PIC X(30).
     05  REC-AMT-CNT           PIC 9(4).
     05  REC-AMT               PIC 9(5)
           OCCURS 1 TO 100 TIMES
             DEPENDING ON REC-AMT-CNT.
     .
     .
PROCEDURE DIVISION.
     .
     .
     EXEC CICS READ DATASET("INFILE")
         RIDFLD(INFILE-KEY)
         SET(RECORD-POINTER)
         LENGTH(RECORD-LEN)
     END-EXEC.
     MOVE REC-AMT-CNT TO REC-AMT-CNT.
     MOVE REC-OTHER-DATA TO WS-DATA.
     MOVE VAR-RECORD TO WS-RECORD-HOLD.
     .
     .
```

COBOL/VSE

```
LINKAGE SECTION.
   01 VAR-RECORD.
      05  REC-OTHER-DATA               PIC X(30).
      05  REC-AMT-CNT                  PIC 9(4).
      05  REC-AMT                      PIC 9(5)
             OCCURS 1 TO 100 TIMES
               DEPENDING ON REC-AMT-CNT.
      .
      .
PROCEDURE DIVISION.
       .
       .
       EXEC CICS READ DATASET("INFILE")
             RIDFLD(INFILE-KEY)
             SET(ADDRESS OF VAR-RECORD)
             LENGTH(RECORD-LEN)
       END-EXEC
       MOVE VAR-RECORD TO WS-RECORD-HOLD
       .
       .
```

4.5.2 DL/I Considerations

LE/VSE provides DL/I support for COBOL/VSE. This section summarizes the migration considerations for DL/I applications.

Subtopics:

4.5.2.1 Interfaces to DL/I

The following DL/I interfaces called from COBOL are supported:

- CBLTDLI
- EXEC DLI
- CEETDLI

LE/VSE provides the callable service CEETDLI that you can use to invoke DL/I. CEETDLI performs basically the same functions as the CBLTDLI and EXEC DLI interfaces.

Please note that CEETDLI works with assembler, COBOL and PL/I programs. This makes CEETDLI language independent. Since the coding of the CEETDLI call is the same in all the mentioned LE/VSE-languages, it is recommended to use CEETDLI instead of the language dependent interfaces.

4.5.2.2 Compile and Link Considerations

For DOS/VS COBOL it was necessary to include DLZBPJRA when link-editing the phase. When the main program is COBOL/VSE, it is not necessary to include DLZBPJRA when linking the phase. The entry point for the phase must be DLITCBL.

4.5.2.3 Condition Handling

The DL/I environment is sensitive to errors or conditions. A failing DL/I transaction or application can contaminate a DL/I database. For this reason, it is essential that DL/I knows about the failure of a transaction or application that has been updating a database so that it can perform necessary clean-up activities.

For a detailed description of DL/I condition handling, see chapter "Running Applications with DL/I" in the *IBM Language Environment for VSE/ESA Programming Guide*.

4.5.3 SQL/DS Considerations

The following special considerations apply to migrating COBOL programs that use SQL/DS.

Subtopics:

4.5.3.1 Using Dynamic SQL in COBOL/VSE

Under DOS/VS COBOL, you can use all forms of the dynamic Structured Query Language (SQL). However, if you want to use an SQL Data Area (SQLDA), you must write an assembler subroutine to manage address variables (pointers) and storage allocation.

Under COBOL/VSE, the assembler subroutine is no longer needed. In addition, non-SELECT statements, fixed-list SELECT statements, and varying-list SELECT statements can also be processed.

For COBOL/VSE, the steps are the same as for assembler and PL/I. However, some of the coding techniques are different. COBOL/VSE uses the following SQL statements:

PREPARE

DESCRIBE

OPEN

CLOSE

FETCH

For more detailed information and examples, see the *SQL/Data System Application Programming for VSE*.

4.5.3.2 Setting Compiler Options with SQL/DS-TRUNC Option

When running COBOL/VSE programs on SQL/DS that use binary data items that may contain a value larger than that defined by the data item's PICTURE clause, it is recommended that you use the TRUNC(BIN) compiler option. (This option gives correct results when COBOL/VSE programs interface with certain products, like SQL/DS, that use standard S/370-format binary data).

TRUNC(BIN) tells the compiler to shorten a binary arithmetic field based on the size of the binary field in storage.

Note: Use of the TRUNC(BIN) compiler option may adversely affect the performance of the application. For more information, see the *COBOL/VSE Programming Guide*.

4.5.3.3 Backing Out Changes During Cutover to Production Mode

During cutover to production mode, you may discover errors in processing that were not detected during testing. If this happens, you should ensure that the erroneous data is not committed. Instead, you should return to the last commit point (with an SQL ROLLBACK WORK statement) and then continue processing from that point using the unmigrated DOS/VS COBOL program.

For a complete discussion, see *SQL/Data System Interactive SQL User's Reference for VSE*.

4.5.3.4 Condition Handling

An SQL/DS database can be contaminated if errors occurring in SQL/DS are not handled properly. For this reason, any errors in SQL/DS must be trapped and handled by SQL/DS. If a task terminates, SQL/DS can then take appropriate action depending on the nature of termination.

For a detailed description of SQL/DS condition handling, see chapter "Running Applications with SQL/DS" in the *IBM Language Environment for VSE/ESA Programming Guide*.

5.0 Adding COBOL/VSE Programs to Existing Applications

Subtopics:

5.1 Chapter 12. Exploiting LE/VSE by Adding COBOL/VSE Programs to Existing Applications

When you add a COBOL/VSE program to an existing application, you are either recompiling an existing program with COBOL/VSE or including a newly-written COBOL/VSE program. When you add COBOL/VSE programs to your existing applications you can receive the following benefits:

- Use a 4-digit year in the date function for new applications.
- Interpret 2-digit year data as 4-digit years for existing applications.
- Continue using user-established error handling routines (modified).
- Upgrade your existing programs incrementally, as your needs dictate.

Restriction: With CICS, you cannot mix DOS/VS COBOL and COBOL/VSE programs in the same run unit.

Once you begin adding COBOL/VSE programs to your existing applications, you need to know the following:

- Link-edit requirements
- Restrictions on running programs above the 16-megabyte line

Subtopics:

5.1.1 Considerations for Link-Editing Existing Applications with LE/VSE

When you begin adding COBOL/VSE programs to existing applications, you must use the PRD2.SCEEBASE sublibrary. Linking to PRD2.SCEEBASE impacts the remaining programs in the applications. How it affects existing applications depends on whether the application includes:

- DOS/VS COBOL programs
- VS COBOL II programs compiled with RES
- VS COBOL II programs compiled with NORES

Subtopics:

5.1.1.1 Adding COBOL/VSE Programs to DOS/VS COBOL Applications

When you add a COBOL/VSE program to an application containing only DOS/VS COBOL programs, you need to

- Link-edit the DOS/VS COBOL program with LE/VSE
- Understand the requirements for using dynamic and static calls

Under LE/VSE on non-CICS, you can issue both static and dynamic calls between DOS/VS COBOL, VS COBOL II, and COBOL/VSE programs. However, there are some restrictions for calls between DOS/VS COBOL and COBOL/VSE

programs.

If you issue static calls with DOS/VS COBOL and COBOL/VSE programs, thereby forming a single phase, the phase must reside below the 16-megabyte line. The phase will be marked RMODE(24), AMODE(24).

If you issue static calls with DOS/VS COBOL and COBOL/VSE programs, thereby forming a single phase, the phase must reside below the 16-megabyte line. The phase will be marked RMODE(24), AMODE(24).

Dynamic calls from VS COBOL II and COBOL/VSE to DOS/VS COBOL programs are supported with the following restrictions:

- The DOS/VS COBOL program must be relinked with the LE/VSE Run-Time Compatibility library.
- DOS/VS COBOL programs that are segmented cannot be loaded into GETVIS storage. These are programs that are compiled with LANGLVL(1) and specify the SEGMENT-LIMIT clause.
- The debug options STATE, FLOW, COUNT and SYMDMP are disabled.

Once you add a COBOL/VSE program to an application with all DOS/VS COBOL programs, you must link-edit every DOS/VS COBOL program in the application with LE/VSE.

Behavior before link-edit with LE/VSE: Before adding the COBOL/VSE program, link-editing an application with all DOS/VS COBOL programs with LE/VSE was not required and, if you did link-edit with LE/VSE, it had no effect.

Behavior after link-edit with LE/VSE: After adding the COBOL/VSE program and link-editing the remaining programs with LE/VSE, the DOS/VS COBOL programs now have access to the LE/VSE default assembler user-exit and installation-wide run-time option CSECT.

Link-Edit Replace Requirement: For phases with both COBOL/VSE and DOS/VS COBOL programs, you must set the default AMODE setting to AMODE(24) when the phase contains a COBOL/VSE program compiled with NORENT that issues static calls to other programs in the application. For instructions on how to replace the default AMODE, see the *COBOL/VSE Programming Guide*.

After adding a COBOL/VSE or VS COBOL II program to an application with all DOS/VS COBOL programs, the following language elements will behave differently:

- UPSI switches may only be set using the LE/VSE UPSI run-time option. UPSI switches set using the VSE JCL UPSI statement will not be honored if a VS COBOL II or COBOL/VSE program is present in the application.

  VS COBOL II and COBOL/VSE compiled programs are able to modify UPSI switch settings. However, the UPSI switch settings available to DOS/VS COBOL compiled programs will be those current at the time the program was invoked. Changes to UPSI switches made by VS COBOL II or COBOL/VSE programs called by a DOS/VS COBOL program will not be available to the DOS/VS COBOL program until it is re-invoked.

- ACCEPT from SYSIN, SYSIPT, or the corresponding mnemonic name as specified in the SPECIAL-NAMES paragraph, will not read past end-of-file on the SYSIPT logical unit. Previously, DOS/VS COBOL compiled programs were able to read multiple files from the SYSIPT logical unit, each file being terminated with a /* card. If a VS COBOL II or COBOL/VSE program is present in the VSE/ESA batch application, only one file may be read. Subsequent ACCEPT statements after end-of-file has been detected will result in the data item specified in the ACCEPT statement being unchanged.

- If a VS COBOL II or COBOL/VSE program is present in the batch application with a DOS/VS COBOL program, the AIXBLD and DEBUG options will be determined by LE/VSE and not from the JCL OPTION SYSPARM. The AIXBLD and DEBUG options can only be specified via the run-time options modules or the PARM parameter of the JCL EXEC statement.

- If a VS COBOL II or COBOL/VSE program is present in the batch application with a DOS/VS COBOL program, the USE AFTER STANDARD ERROR declarative in DOS/VS COBOL will only be effective if the TRAP(OFF) option has been specified.

- If a SORT or MERGE statement is executed from a DOS/VS COBOL program, and the SORT/MERGE product issues a STXIT AB or STXIT PC, and does not reset the STXIT AB and/or STXIT PC upon return, the TRAP(ON) option will be disabled.

- A STOP RUN from a DOS/VS COBOL program will result in control being returned to the caller of the main program, instead of terminating via an EOJ.

If DOS/VS COBOL programs are loaded into program storage via an assembler program, the debug options STATE, FLOW, COUNT and SYMDMP will be disabled for the second or subsequent phase that contains a DOS/VS COBOL program.

A DOS/VS COBOL program may not load a VS COBOL II or COBOL/VSE program using an assembler program, unless a VS COBOL II or COBOL/VSE program is link-edited into the phase containing the DOS/VS COBOL program.

DOS/VS COBOL programs combined with VS COBOL II RES programs, will need to be relinked to run under COBOL/VSE.

5.1.2 Adding COBOL/VSE Programs to VS COBOL II Application Compiled RES

If you add a COBOL/VSE program to an application with all VS COBOL II RES programs, the application is unchanged and does not need to be relinked.

5.1.3 Adding COBOL/VSE Programs to VS COBOL II Application Compiled NORES

Once you add a COBOL/VSE program to an application with all NORES

programs, you must link-edit every NORES program in the application with LE/VSE. You now have additional considerations for these NORES programs.

Behavior before Link-Edit with LE/VSE: Before adding the COBOL/VSE program, link-editing an application with all NORES programs with LE/VSE was not required.

Behavior after Link-Edit with LE/VSE: After adding the COBOL/VSE program and link-editing the remaining programs with LE/VSE, the NORES programs now have access to the LE/VSE default assembler user exit and the installation-wide run-time options CSECT.

More importantly, these NORES programs are now 'RES-like'. All of the considerations for programs compiled RES now apply to these NORES programs that you have link-edited with LE/VSE.

APPENDIX1 Appendixes

Subtopics:

APPENDIX1.1 Appendix A. Commonly Asked Questions and Answers

This section provides answers to some of the most common questions people have about upgrading to COBOL/VSE and LE/VSE. The questions are grouped into the following categories:

- Prerequisites
- Compatibility
- Link-Editing with LE/VSE
- Compiling with COBOL/VSE

- LE/VSE Services
- LE/VSE Run-Time Options
- Interlanguage Communication
- Subsystems

Subtopics:

APPENDIX1.1.1 Prerequisites

ÿ ***Is LE/VSE required to run a program compiled with COBOL/VSE?***

Yes, you can only run programs compiled with COBOL/VSE with LE/VSE. LE/VSE contains the library routines required to run a COBOL/VSE compiled program. COBOL/VSE is a compiler and does not contain any run-time library routines.

ÿ ***Why does LE/VSE require CICS/VSE Version 2 Release 3 ?***

Because changes to CICS were required to support LE/VSE.

ÿ ***Can you use LE/VSE with versions of CICS prior to Version 2 Release 3 if the applications being run are VS COBOL II or DOS/VS COBOL applications?***

No. LE/VSE requires CICS Version 2 Release 3 even if the applications were created using previous versions of CICS and COBOL.

ÿ ***Do you have to convert all COBOL programs to CICS Version 2 Release 3 before they can be run with LE/VSE?***

Yes. You can only run LE/VSE in CICS Version 2 Release 3, so any programs that you want to run under LE/VSE must be upgraded to CICS Version 2 Release 3.

APPENDIX1.1.2 Compatibility

ÿ ***Can you run DOS/VS COBOL programs with LE/VSE?***

Yes. DOS/VS COBOL, VS COBOL II, and COBOL/VSE programs are all supported.

ÿ ***When running DOS/VS COBOL in compatibility mode with VS COBOL II or LE/VSE, are the run-time control blocks accessed the same way?***

Yes. The pointers in the TGT and other control blocks can still be used to get to the control blocks for DOS/VS COBOL.

ÿ ***When should you use the CMPR2 option?***

Use the CMPR2 option when a program written for VS COBOL II Release 2 does not produce the expected results under NOCMPR2, or does not compile successfully due to changes in language elements. You can also use CMPR2 when compiling DOS/VS COBOL programs that meet the COBOL 74 Standard and do not use any unsupported language elements or invalid language.

Note: NOCMPR2, supporting the COBOL 85 Standard, is the strategic standard for IBM COBOL compiler.

ÿ ***Can you place LE/VSE in the permanent LIBDEF chain?***

Before placing LE/VSE in the permanent LIBDEF chain, test all applications that might access the LE/VSE library routines from the permanent LIBDEF chain. The move to LE/VSE should be staged in a controlled manner.

ÿ ***Can LE/VSE coexist on a system with DOS/VS COBOL and VS COBOL II?***

Yes, but be aware of which sublibrary is needed and used for different applications. There are duplicate names between the different product sublibraries, so it is important to ensure the correct sublibrary is being accessed.

ÿ ***Is it easier to convert from DOS/VS COBOL to COBOL/VSE or from DOS/VS COBOL to VS COBOL II?***

The source conversion effort is exactly the same. Moving to the LE/VSE run time is slightly more difficult if you have assembler programs that use condition handling with DOS/VS COBOL. Since the majority of time is spent with testing, the two different conversion paths are about the same.

ÿ ***Have there been any statements about when either support or marketing of VS COBOL II will be discontinued?***

There has been no announcement of service withdrawal for VS COBOL II Release 4. Level II service for VS COBOL II Release 4 is available at no charge.

ÿ ***Is the signature area of COBOL/VSE programs the same as for DOS/VS COBOL and VS COBOL II?***

No, but a map of the signature area is in the *COBOL/VSE Programming Guide*. This can be used to find out what compiler options were used to compile the module, when it was compiled, release level, etc.

ÿ ***Is it true that converting from DOS/VS COBOL to COBOL/VSE gives a 20% saving?***

No, this is not true. COBOL/VSE is comparable or a little bit slower than DOS/VS COBOL.

APPENDIX1.1.3 Link-Editing with LE/VSE

ÿ ***When is it necessary to link-edit applications with LE/VSE in order to run under LE/VSE?***

See:

- "Determine which DOS/VS COBOL Programs Require Link-Edit with LE/VSE" in topic 3.2.1 for programs run under the DOS/VS COBOL library.
- "Determine which VS COBOL II Programs Require Link-Edit with LE/VSE" in topic 3.3.1 for programs run under the VS COBOL II library.

ÿ ***Can DOS/VS COBOL and VS COBOL II programs call COBOL/VSE programs?***

Yes. On non-CICS, any CALLs between DOS/VS COBOL, VS COBOL II, and COBOL/VSE are supported. For programs running under CICS, the following applies: VS COBOL II and COBOL/VSE programs can issue any calls between each other. However, no calls can be made to DOS/VS COBOL programs, and DOS/VS COBOL programs cannot call a VS COBOL II or COBOL/VSE program.

You must link-edit the entire application with LE/VSE if any program within the application is either compiled with COBOL/VSE or link-edited with LE/VSE. A mixture of library

routines from different sublibraries is not allowed.

ÿ ***Can DOS/VS COBOL phases CALL and be CALLed by COBOL/VSE?***

COBOL/VSE phases can call DOS/VS COBOL phases, if the DOS/VS COBOL phase has been link-edited with LE/VSE. The DOS/VS COBOL phase must return control to the COBOL/VSE phase.

It is possible to have a DOS/VS COBOL phase statically calling assembler, which dynamically calls DOS/VS COBOL. This type of phase cannot call COBOL/VSE. Instead, replace the call to the assembler program with a call to a COBOL/VSE routine, which can then call subsequent routines.

ÿ ***Can you convert programs selectively to COBOL/VSE?***

For non-CICS applications, yes, as long as you follow the rules for link-editing (documented throughout this book).

For CICS applications, you cannot mix DOS/VS COBOL programs and COBOL/VSE programs in the same run unit. When converting applications containing DOS/VS COBOL programs that are run under CICS, you must convert all the DOS/VS COBOL programs to COBOL/VSE.

APPENDIX1.1.4 Compiling with COBOL/VSE

ÿ ***Can you compile programs written for DOS/VS COBOL with COBOL/VSE using the CMPR2 option?***

Some DOS/VS COBOL source programs might compile cleanly under COBOL/VSE with the CMPR2 option. The CMPR2 option supports COBOL 74 Standard as implemented by VS COBOL II Release 2. DOS/VS COBOL LANGLVL(2) programs might compile if they do not use any unsupported language elements or invalid language which might have run in DOS/VS COBOL.

ÿ ***Can you compile programs written for VS COBOL II with COBOL/VSE?***

Yes. Programs written for VS COBOL II will compile with few or no problems under COBOL/VSE. There are two new reserved words, FUNCTION and PROCEDURE-POINTER, and three interpretation changes of COBOL 85 Standard elements. They are:

1. Reference-modified variable-length group receivers that contain their own OCCURS DEPENDING ON objects

2. Comment lines in matching REPLACE text (not COPY REPLACING)

3. GLOBAL error declaratives in nested programs

These three language elements cannot exist in any current COBOL programs; however, they are flagged by COBOL/VSE.

ÿ ***Do you have to compile DOS/VS COBOL and VS COBOL II programs with COBOL/VSE to run them under LE/VSE?***

No. Most DOS/VS COBOL programs and VS COBOL II programs (and mixes of the two) will run under LE/VSE without the need to compile with COBOL/VSE or link-edit with LE/VSE. You can use LE/VSE as the run-time library for DOS/VS COBOL programs and VS COBOL II programs compiled with the RES compiler option.

There are two cases when you have to compile with COBOL/VSE:

1. If an application is **all** of the following:

   - Consists of RES and NORES programs and
   - Runs with the MIXRES run-time option under VS COBOL II and
   - Is NOT linked with IGZBRDGE and
   - Is either a single phase application or is the first phase in an application.

2. If an application has DOS/VS COBOL programs that call PL/I or are called by PL/I.

ÿ ***What utilities or tools can assist in converting DOS/VS COBOL source to COBOL/VSE source?***

There are two conversion aids that assist in converting DOS/VS COBOL source to COBOL/VSE source.

1. COBOL and CICS Command Level Conversion Aid for VSE, 5785-CCC, (CCCA) assists in converting DOS/VS COBOL source to COBOL/VSE source.

2. The Report Writer Precompiler, 5798-DYR, assists with converting DOS/VS COBOL Report Writer code.

COBOL Structuring Facility (COBOL/SF), assisting with the conversion process by re-engineering existing COBOL code into structured code. It does **not** convert DOS/VS COBOL code into COBOL/VSE code.

ÿ ***Does COBOL/VSE meet the COBOL 85 Standard?***

Yes, COBOL/VSE supports all required modules of the COBOL 85 Standard at most of the highest level defined by the Standard. In addition, some of the optional modules are supported, including the Intrinsic Functions Module.

The industry standards COBOL/VSE supports, and the exceptions, are described in Appendix I, "Industry Standards" in topic APPENDIX1.9 and in *COBOL/VSE Licensed Program Specifications*.

APPENDIX1.1.5 LE/VSE Services

ÿ ***Can VS COBOL II programs call LE/VSE service routines?***

VS COBOL II programs can issue dynamic calls to LE/VSE service routines, but not static calls.

ÿ ***Can users call LE/VSE callable services from BAL (Assembler) programs if the programs are LE/VSE-enabled assembler programs?***

Yes. Any LE/VSE-enabled BAL routines can use LE/VSE callable services. Non-LE/VSE-enabled BAL routines **cannot** use LE/VSE callable services. The *LE/VSE Programming Guide* describes how to make your existing BAL programs into LE/VSE-enabled BAL programs, using macros supplied by IBM with the LE/VSE product.

ÿ ***Can DOS/VS COBOL programs use LE/VSE callable services?***

No. DOS/VS COBOL programs cannot use the LE/VSE callable services directly, but DOS/VS COBOL programs can call a COBOL/VSE program to place calls to the services.

ÿ ***Can exception handling be added to a pure DOS/VS COBOL application by converting the main routine to COBOL/VSE?***

Yes, with restrictions on recovery. With DOS/VS COBOL, you cannot move the resume cursor from the routine where the error occurred if that routine will be re-entered without an intervening CANCEL. This is because the routine will think it is being entered recursively. After condition handling for an error in a DOS/VS COBOL routine, you must resume in that same routine.

APPENDIX1.1.6 LE/VSE Options

ÿ ***Will lower HEAP storage values for COBOL performance affect PL/I performance?***

In general, the answer is no. However, performance might be slower for applications that have a high use of ALLOCATE and FREE. In this case, you should tune the HEAP values to improve performance. Also, if the application has many static variables, the STACK values should also be tuned to improve performance.

ÿ ***Does COBOL use STACK storage?***

COBOL programs do not use STACK storage. COBOL run-time routines do use STACK storage.

ÿ ***Does DOS/VS COBOL running on LE/VSE use HEAP storage?***

No, DOS/VS COBOL working storage does not use HEAP storage.

ÿ ***What does HEAP(KEEP) or LIBSTACK(KEEP) do? Does it keep all of the HEAP or LIBSTACK storage or just the increments of extra storage that were obtained?***

The KEEP suboption causes LE/VSE to keep **all** of the storage obtained, including the initial and incremental amounts.

ÿ ***How does ERRCOUNT relate to abends? Does ERRCOUNT only count HANDLED conditions?***

ERRCOUNT is a count of errors/conditions/abends/exceptions that are allowed before LE/VSE abends with its own abend code. If an error is not HANDLED the application will terminate so ERRCOUNT will have no effect.

APPENDIX1.1.7 Interlanguage Communication

ÿ ***Why are assembler language programs not discussed under interlanguage communication in LE/VSE manuals or in IBM presentations?***

Assembler language is not considered a separate language by LE/VSE. There is no run-time library associated with assembler programs, so there are no run-time conflicts with assemblers and other High-Level Languages (HLLs). Assembler programs can call and be called by any of the HLLs supported by LE/VSE.

ÿ ***Can DOS/VS COBOL or VS COBOL II programs call LE/VSE-enabled assembler programs?***

Only dynamic calls between VS COBOL II and LE/VSE-enabled assembler are supported.

APPENDIX1.1.8 Subsystems

ÿ ***Does COBOL/VSE have a method to return the century information from the system as CICS does? We need this for proper handling in SQL/DS.***

COBOL/VSE and LE/VSE each provide a method in which to handle the year 2000 issue. With COBOL/VSE, you can use the CURRENT-DATE intrinsic function. With LE/VSE, you can use the DATE callable service.

ÿ ***When running in a CICS region, does EXEC DLI "translate" into interfacing with CEETDLI or CBLTDLI?***

EXEC DLI does not "translate" into interfacing with either CEETDLI or CBLTDLI. The CICS translator generates a call to DFHELI. The call to DFHELI must be a static call. (The NODYNAM compiler option is required for programs translated by the CICS translator.)

ÿ ***Is CALL 'CEETDLI' supported in a CICS program? What about CALL 'CBLTDLI' in a CICS program running under LE/VSE?***

CEETDLI is not supported under a CICS environment (CICS does not supply a CEETDLI entry point in DFHDLIAL). CBLTDLI is supported under a CICS environment (CICS does supply a CBLTDLI entry point in DFHDLIAL) under LE/VSE.

ÿ ***If you have a batch application that has explicit calls to other LE/VSE services, or user-coded LE/VSE condition handlers, must all interfaces use CEETDLI instead of CBLTDLI?***

No, all calls within a program or run unit are not required to be CEETDLI.

ÿ ***Will LE/VSE (and its support of mixed COBOL and PL/I programs) still support mixed PL/I and VS COBOL II (or COBOL/VSE) applications where the COBOL programs use CBLTDLI - or must such programs be converted to CEETDLI?***

There is no problem with a mixed environment from a DL/I standpoint and the programs do not need to be modified. Consider CBLTDLI and CEETDLI equivalent for conversion purposes.

Under LE/VSE, your COBOL programs can still use the CBLTDLI interface. Remember the programs must be VS COBOL II or COBOL/VSE, since mixed DOS/VS COBOL and PL/I is not allowed under LE/VSE. Either CBLTDLI or CEETDLI can be used, except that CEETDLI is not supported under a CICS environment.

ÿ ***Is TRAP(ON) supported in DL/I when using CBLTDLI?***

Yes. However, when CBLTDLI is used with TRAP(ON) and an error occurs in DL/I, the LE/VSE assembler user exit is driven for termination processing.

APPENDIX1.2 Appendix B. Conversion Aids for Source Programs

This appendix describes the conversion aids available for your assistance during the actual conversion activity. These aids are:

- COBOL/VSE conversion aids
- DOS/VS COBOL conversion aids
- Other programs that aid conversion

After reading this appendix, you should be able to determine which, if any, of the conversion aids to use. You should be able to understand how to use them, and how to analyze the conversion aid output to assess the extent of the remaining conversion effort.

Note: All of these conversion aids assume that the program you are converting is a valid DOS/VS COBOL or VS COBOL II program; that is, it is a program that is written according to the rules given in *IBM VS COBOL for DOS/VSE* and *VS COBOL II Application Programming: Language Reference*, with no undocumented extensions.

Subtopics:

APPENDIX1.2.1 COBOL/VSE Conversion Aids - CMPR2, FLAGMIG, and NOCOMPILE Compiler Options

By compiling existing DOS/VS COBOL application programs with the COBOL/VSE compiler, you can identify some of the DOS/VS COBOL and VS COBOL II source language that needs modification.

CMPR2 Generates code that provides COBOL 74 Standard behavior, as implemented by VS COBOL II Release 2 under MVS and VM, as well as nonstandard Release 2 extensions now implemented in the COBOL 85 Standard.

FLAGMIG Identifies DOS/VS COBOL and VS COBOL II language elements that are incompatible with COBOL/VSE NOCMPR2.

Note: Implementation of the COBOL 85 Standard created some instances where incompatibilities with VS COBOL II Release 2 can occur. Use of the CMPR2 and FLAGMIG compiler options aid in the conversion of programs to COBOL/VSE NOCMPR2.

NOCOMPILE Identifies any language elements not accepted by the COBOL/VSE compiler without producing an object deck, thus saving system resources over a full compile with the COMPILE compiler option.

APPENDIX1.2.2 DOS/VS COBOL Conversion Aid--MIGR Compiler Option

You can use the DOS/VS COBOL MIGR compiler option whenever you are planning to convert a DOS/VS COBOL program to COBOL/VSE with NOCMPR2. This option helps you understand the magnitude of the conversion effort. It can also ease any planned future conversion, by helping you avoid using DOS/VS COBOL source language not supported by COBOL/VSE with NOCMPR2. By compiling your programs using MIGR, you can determine ahead of time what language elements must be converted.

There are incompatibilities in the following areas:

- New reserved words introduced for added COBOL functions. (Previously valid user words might now be invalid)
- Language function supported in a different manner
- Language function no longer supported

You can set the MIGR compiler option either as an installation default at install time, or when compiling a DOS/VS COBOL program. When you set MIGR on, the compiler flags most statements that are changed in, or not supported by, COBOL/VSE with NOCMPR2.

The MIGR option **does not** flag the following:

- Undocumented extensions to DOS/VS COBOL. (Some of these extensions are described in "Undocumented DOS/VS COBOL Extensions" in topic 4.1.5.)

Subtopics:

APPENDIX1.2.2.1 Language Differences between COBOL/VSE and DOS/VS COBOL

- ALPHABETIC Class Changes

- B Symbol in PICTURE Clause
- CALL Statement Changes
- Combined Abbreviated Relation Condition Changes
- COM-REG special register
- EXIT PROGRAM (or STOP RUN) missing at program end
- FILE STATUS clause
- ID1 IS [NOT] ALPHABETIC

  (class test on IF, PERFORM, and SEARCH)

- IF...OTHERWISE Statement Changes
- MOVE A TO B

  where B is defined as a variable length data item containing its own ODO object

- MULTIPLY|DIVIDE ID1 BY|INTO ID2 [GIVING ID3] ON SIZE ERROR ...
- NSTD-REELS special register
- PERFORM P1 [THRU P2] VARYING ID2 FROM ID3 BY ID4 UNTIL COND-1
  AFTER ID5 FROM ID6 BY ID7 UNTIL COND-2
  AFTER ID8 FROM ID9 BY ID10 UNTIL COND-3

  1. Where ID6 is (potentially) dependent on ID-2
  2. Where ID9 is (potentially) dependent on ID-5
  3. Where ID4 is (potentially) dependent on ID-5
  4. Where ID7 is (potentially) dependent on ID-8

  Dependencies occur when the first identifier or index name (IDx) is identical to, subscripted with, or qualified with the second identifier. Dependencies might also occur with a partial or full redefinition of the second identifier.

- OCCURS DEPENDING ON Clause Changes
- ON SIZE ERROR Option--Changes in Intermediate Results

- PROGRAM COLLATING SEQUENCE Clause Changes
- READ filename RECORD INTO B

  where B is defined variable length data containing the object of the ODO phrase

- RECORD CONTAINS integer-4 CHARACTERS in the FD Section
- RERUN Clause Changes
- RESERVE Clause Changes
- Reserved Word List Changes
- SEARCH or SEARCH ALL...
- SERVICE RELOAD option
- SORT-OPTION IS clause
- SPECIAL-NAMES: alphabet-name IS xxxxx
- Subscripts Out of Range--flagged at Compile-Time
- UNSTRING A INTO B ...

  where B is defined variable length data containing the object of the ODO phrase

- UNSTRING ID1 DELIMITED BY ID2 INTO ID4 DELIMITER IN ID5 COUNT IN ID6 WITH POINTER ID7
- UPSI and UPSI mnemonic names references
- VALUE Clause Condition Names
- WHEN-COMPILED Special Register
- WRITE BEFORE/AFTER ADVANCING PAGE Statement
- WRITE AFTER POSITIONING

APPENDIX1.2.2.2 Statements Supported with Enhanced Accuracy in COBOL/VSE

Following are DOS/VS COBOL statements supported with enhanced accuracy in

COBOL/VSE and flagged by a message indicating that more accurate results might be provided in COBOL/VSE.

Arithmetic Statements

- Definitions of floating-point data items
- Use of floating-point literals
- Use of exponentiation

APPENDIX1.2.2.3 LANGLVL(1) Statements Not Supported in COBOL/VSE

The following DOS/VS COBOL statements, applicable only to the LANGLVL(1) compiler option, are not supported in COBOL/VSE and are flagged when the MIGR compiler option is specified.

- COPY language--1968
- JUSTIFIED|JUST clause with VALUE
- NOT in an Abbreviated Combined Relation Condition
- RESERVE integer AREAS
- SELECT OPTIONAL clause--1968 Standard interpretation
- SPECIAL-NAMES paragraph: use of L, /, and =
- UNSTRING with DELIMITED BY ALL

APPENDIX1.2.2.4 LANGLVL(1) and LANGLVL(2) Statements Not Supported in COBOL/VSE

The following DOS/VS COBOL statements, applicable to both the LANGLVL(1) and LANGLVL(2) compiler options, are not supported in COBOL/VSE and are flagged when the MIGR compiler option is specified.

Report Writer

- INITIATE, GENERATE, and TERMINATE verbs
- LINE-COUNTER, PAGE-COUNTER, and PRINT-SWITCH special registers
- Nonnumeric literal IS mnemonic-name in SPECIAL NAMES
- REPORT clause of FD
- REPORT SECTION Header
- USE BEFORE REPORTING declarative

Note: The Report Writer Precompiler Program Offering can convert these statements for you. See "COBOL Report Writer Precompiler" in topic APPENDIX1.2.3.2.

ISAM

- APPLY REORG-CRITERIA (ISAM)
- APPLY CORE-INDEX (ISAM)
- I/O verbs--all that reference ISAM files
- ISAM file declarations
- NOMINAL KEY clause
- Organization parameter "I"

- TRACK-AREA clause
- USING KEY clause on START statement

DAM

- ACTUAL KEY clause
- APPLY RECORD-OVERFLOW (DAM)
- DAM file declarations
- I/O verbs--all that reference DAM files
- Organization parameters "D", "R", and "W"
- SEEK statement
- TRACK-LIMIT clause

Other Statements

- APPLY RECORD-OVERFLOW
- Assignment-name organization parameter "C" indicating ASCII
- ASSIGN TO integer system-name
- ASSIGN ... FOR MULTIPLE REEL/UNIT
- CURRENT-DATE and TIME-OF-DAY special registers
- Debug packets
- EXAMINE statement
- EXHIBIT statement
- FILE-LIMITS
- LABEL RECORDS is data-name on non-sequential files
- NOTE statement
- ON statement
- Qualified index-names

  (Using this unsupported format will generate an S-level message in COBOL/VSE.)
- READY TRACE and RESET TRACE Statements
- REMARKS paragraph
- RESERVE NO/ALTERNATE AREAS

- SEARCH...WHEN condition using KEY item as object, not subject
- SERVICE RELOAD statement
- START...USING Key statement
- THEN as a statement connector
- TIME-OF-DAY Special Register
- TRANSFORM statement
- USE AFTER STANDARD ERROR ... GIVING
- USE BEFORE STANDARD LABEL
- USING procedure-name or file-name on CALL statement

APPENDIX1.2.3 Other Programs That Aid Conversion

The following sections describe several program products and program offerings that offer you help in your conversion tasks. These programs are:

- COBOL and CICS Command Level Conversion Aid for VSE (CCCA)
- COBOL Report Writer Precompiler
- COBOL Structuring Facility (COBOL/SF)

Subtopics:

APPENDIX1.2.3.1 COBOL and CICS Command Level Conversion Aid for VSE (CCCA)

The COBOL and CICS Command Level Conversion Aid for VSE (CCCA) Program Offering, Product Number 5785-CCC, converts CICS and non-CICS source code into COBOL/VSE source code.

CCCA is designed to automate identifying incompatible source code and converting it to COBOL/VSE source. Using CCCA should significantly reduce your conversion effort.

CCCA requires that you have the COBOL/VSE and the DOS/VS COBOL and/or VS COBOL II compilers available when converting CICS programs.

The following are the key CCCA facilities:

- Conversion of most syntax differences between DOS/VS COBOL and VS COBOL II programs and COBOL/VSE programs.
- Elimination of conflicts between DOS/VS COBOL and VS COBOL II user-defined names and COBOL/VSE reserved words.
- Flagging of language elements that cannot be directly converted.
- Statement-by-statement diagnostic listing.
- Conversion management information, including where-used reports for COPY books and files.
- Conversion of EXEC CICS commands.
- Removal and/or conversion of the Base Locator for Linkage Section (BLL) mechanism and references.

CCCA is designed so that you can tailor it to fit the needs of your site. The CCCA Language Conversion Programs (LCPs), which determine the conversions to be performed, are written in a COBOL-like language. You can modify the supplied LCPs or add your own.

For more detail, see the *COBOL and CICS Command Level Conversion Aid for VSE* manual.

Note: If you need to both convert your programs and structure your code and you have COBOL/SF available, you can select an option within COBOL/SF that will activate CCCA. CCCA will convert your program; then, COBOL/SF will structure it.

Subtopics:

APPENDIX1.2.3.1.1 When to Use CCCA

If you plan to convert your applications from DOS/VS COBOL or VS COBOL II to COBOL/VSE, you should evaluate the usefulness of CCCA to your conversion project. While the number of changes required to any individual program might be small, CCCA will identify those changes, and in the majority of cases convert them automatically in a standard fashion. CCCA converts both CICS and non-CICS programs. CCCA converts the complicated logic of BLL cell addressing and SERVICE RELOAD statements to valid COBOL/VSE statements, as well as handling non-CICS syntax.

APPENDIX1.2.3.1.2 CCCA Processing of CICS Statements

If the CICS option is ON, the BLL definitions and SERVICE RELOAD statements are removed. If the entire BLL structure is redefined, the redefined structure is removed. If the BLLs are not defined with a length of 4 bytes, the CICS conversion cannot be performed.

If needed by the conversion of statements involving the primary BLLs, the following code is generated in the WORKING-STORAGE SECTION for use with the POINTER facility:

```
77  LCP-WS-ADDR-COMP PIC S9(8) COMP.
77  LCP-WS-ADDR-PNTR REDEFINES LCP-WS-ADDR-COMP USAGE POINTER.
```

EXEC CICS Processing: The primary BLLs used with SET options are replaced by the corresponding ADDRESS OF special register. For example:

```
EXEC CICS READ ... SET(BLL1) ...
```

is replaced by:

```
EXEC CICS READ ... SET(ADDRESS OF REC1) ...
```

The statements involved are:

CONVERSE	GETMAIN	ISSUE RECEIVE
LOAD	POST	READ
READNEXT	READPREV	READQ
RECEIVE	RETRIEVE	SEND CONTROL
SEND PAGE	SEND TEXT	

The primary BLLs used with CICS ADDRESS statements are replaced by the corresponding COBOL/VSE ADDRESS OF special register.

For example:

```
EXEC CICS TWA(BLL).
```

is replaced by:

```
EXEC CICS TWA(ADDRESS OF TWA).
```

The options involved are: CSA, CWA, EIB, TCTUA, and TWA.

APPENDIX1.2.3.1.3 Statements Dealing with the Primary BLLs

The statements dealing with the primary BLLs are shown in Figure 34.

Statements dealing with the secondary BLLs are replaced by CONTINUE.

Figure 34. COBOL Statements Dealing with Primary BLLs

Original Source	Source After Conversion
MOVE BLL1 TO BLL2	SET ADDRESS OF REC2 TO ADDRESS OF REC1
MOVE ID TO BLL	MOVE ID TO LCP-WS-ADDR-COMP SET ADDRESS OF REC1 TO LCP-WS-ADDR-PNTR
MOVE BLL TO ID	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC MOVE LCP-WS-ADDR-COMP TO ID
ADD ID1, .. TO BLL	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC ADD ID1, TO LCP-WS-ADDR-COMP SET ADDRESS OF REC TO LCP-WS-ADDR-PNTR
ADD BLL TO ID1, ID2	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC ADD LCP-WS-ADDR-COMP TO ID1, ID2
ADD ID1, ID2 GIVING BLL	ADD ID1, ID2 GIVING LCP-WS-ADDR-COMP SET ADDRESS OF REC TO LCP-WS-ADDR-PNTR
ADD ID, BLL1 GIVING BLL2 BLL3	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC ADD ID, LCP-WS-ADDR-COMP GIVING LCP-WS-ADDR-COMP SET ADDRESS OF REC2 TO LCP-WS-ADDR-PNTR SET ADDRESS OF REC3 TO LCP-WS-ADDR-PNTR
ADD ID1, BLL1 GIVING ID2 ID3	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC ADD ID1, LCP-WS-ADDR-COMP GIVING ID2 ID3
SUBTRACT statements	The conversion is performed in the same way as ADD.
COMPUTE BLL = exp (BLL)	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC COMPUTE LCP-WS-ADDR-COMP = exp (LCP-WS-ADDR-COMP)
COMPUTE ID = exp (BLL)	SET LCP-WS-ADDR-PNTR TO ADDRESS OF REC COMPUTE ID = exp (LCP-WS-ADDR-COMP)
COMPUTE BLL = exp ...	COMPUTE LCP-WS-ADDR-COMP = exp ...

APPENDIX1.2.3.2 COBOL Report Writer Precompiler

This program offering has two functions. It can be used to precompile applications containing Report Writer statements so the code will be acceptable to the COBOL/VSE compiler, or it can permanently convert Report Writer statements to valid COBOL/VSE statements.

The Report Writer precompiler offers the following features:

- Extended Report Writer language capabilities
- Automatic invocation of the target COBOL compiler--as though Report Writer statements in the source program are being processed by the COBOL compiler itself

- Single consolidated source listing merges information from the precompiler listing and the COBOL compiler listings

- COPY library members can contain Report Writer statements

- Supports the COBOL/VSE nested COPY feature

- Performs a diagnostic check of the input Report Writer source statements

- Can be run in stand-alone mode to convert Report Writer statements in your COBOL programs into non-Report Writer COBOL source statements acceptable to the COBOL/VSE compiler

- VSE/ESA, MVS/ESA and VM/ESA Support

For more detail, see *COBOL Report Writer Precompiler Programmer's Manual* and *COBOL Report Writer Precompiler Installation and Operation*.

APPENDIX1.2.3.3 COBOL Structuring Facility (COBOL/SF)

The COBOL Structuring Facility (COBOL/SF) Program Product, Product Number 5688-045, offers you a way to re-engineer your unstructured COBOL programs into structured COBOL code. By automating this process, you can reduce the cost of program maintenance and increase programmer productivity. This helps you protect your investments in existing COBOL programs.

Note: Although COBOL/SF is not available for execution under VSE, you can still benefit from the features it offers if your site operates either a VM or MVS system. When installed under VM or MVS, COBOL/SF can be used to restructure COBOL programs.

The input to COBOL/SF should be valid DOS/VS COBOL, VS COBOL II, or COBOL/VSE source code.

COBOL/SF can also help you establish and enforce structured programming standards for new application program development. You can process all newly developed code through COBOL/SF to ensure that your structured code standards are being maintained.

COBOL/SF eliminates any code that is improper and unstructured. By eliminating ALTERs, GO TOs, and code fall-throughs, it reduces the potential for program error and provides quality code for program maintenance. The COBOL/SF code transformations remove undesirable features from the input source code--features that resulted from poor programming practices or from attempts to recover from the effects of these practices.

COBOL/SF can also reveal design structures that have been lost, due to years of maintenance additions and to program logic changes.

COBOL/SF can sometimes uncover previously unknown information about the input program--information that might provide an entirely different view of how the program works.

The COBOL/SF report documents potential anomalies in the code and in each case describes the action taken by COBOL/SF. The report also details complex parts of the unstructured program. There are other reports available that do the following:

- Sequence and list the input program
- List the output program
- Display cross-reference information from the input program to the output program

COBOL/SF accepts its own output as input. This means that you can re-engineer programs repeatedly, to ensure that their structured code does not deteriorate during maintenance operations, and that the structuring standard previously established by COBOL/SF is maintained. COBOL/SF runs under MVS/ESA and VM/ESA and can process any of the following: Report Writer statements, EXEC SQL, DL/I, or CICS commands. (There might be restrictions when using COBOL/SF with certain CICS commands. See the *COBOL Structuring Facility User's Guide and Reference* manual for information on these restrictions.)

Note: If you need to both convert your programs and structure your code and you have CCCA available, you can select an option within COBOL/SF that will activate CCCA. CCCA will convert your program; then, COBOL/SF will structure it.

APPENDIX1.3 Appendix C. Mixed COBOL and Assembler Applications

This chapter contains information for applications that contain a mix of COBOL programs and assembler programs. It is organized based on running existing converted programs under LE/VSE. This chapter includes information on the following characteristics of mixed COBOL and assembler applications:

- NORES applications that call an assembler program that does a LOAD/BALR to other NORES applications
- A COBOL program that is called by an assembler routine with a LOAD / Branch
- Assembler programs that issue a STXIT AB/STXIT PC
- A COBOL program that calls assembler routines that use high-order byte of Register 13
- Programs that are link-edited using ILBDSET0 or IGZERRE with the assembler driver, when the assembler program does and does not LOAD the routines
- Assembler programs that are coded based on the internals of the ILBD routines

- Assembler programs that do not follow normal save area conventions
- Assembler programs that are LE/VSE-enabled

Subtopics:

APPENDIX1.3.1 Running Existing Mixed COBOL and Assembler Applications under LE/VSE

Subtopics:

APPENDIX1.3.1.1 Calling and Called Assembler Programs

When assembler programs are either called by, or call COBOL programs, they must follow the S/370* linkage convention. If not, your applications will terminate with 40XX abends.

Calling Assembler Programs: When the assembler program is the caller, R13 must point to the caller's register save area (18 words), and the first word of the save area must be zero. The save area back chain must be set.

If the program passes identifiers, a parameter list must be prepared, and the address of this list loaded into R1. R1 must be set to zero if no parameter list is passed. R14 must contain the return address in the assembler program, and R15 must contain the address of the entry point of the COBOL program.

Note: If you have a parameter list, it must be a group of one or more contiguous fullwords, each of which contains the address of a data item to be passed to the COBOL program. It is recommended that the high-order bit of the last fullword address be set to 1, to flag the end of the list.

Called Assembler Programs: A called assembler program must save the registers and store other information in the save area passed to it by the COBOL program. In particular, the COBOL save area must be properly back chained from the save area of an assembler program. The assembler program must also contain a return routine that:

- Loads the address of the save area back into R13
- Restores the contents of the other registers
- Optionally sets a return code in R15
- Branches to the address in R14

APPENDIX1.3.2 Running Modified COBOL Programs and Assembler Programs under LE/VSE

Subtopics:

APPENDIX1.3.2.1 Application with an Assembler Driver and COBOL Subroutines

There are two methods you can use to migrate applications that use a non-COBOL driver to call COBOL subroutines:

1. Modify the driver to set up the LE/VSE environment.
2. Use the RTEREUS run-time option if the driver cannot be modified.

Both of these methods are described in the sections below. (In both methods, you migrate the COBOL subroutines in the same ways as described in the other COBOL conversion scenarios.)

Subtopics:

APPENDIX1.3.2.1.1 Modify Non-COBOL Driver

If you want to modify the non-COBOL driver routine, you can replace the DOS/VS COBOL ILBDSET0 routine with the LE/VSE IGZERRE INIT and IGZERRE TERM functions. These LE/VSE routines have a convenient complementary termination function not available with DOS/VS COBOL.

If either IGZERRE or ILBDSET0 is statically linked with the assembler driver, the assembler driver must be link-edited with LE/VSE. If ILBDSET0 is used, you must specify the LE/VSE options: ALL31(OFF) and STACK(,,BELOW).

For an alternative method using pre-initialization, see "Initialize the Run-Time Environment" in topic 3.3.6.

APPENDIX1.3.2.1.2 Use an Unmodified Driver

If you cannot (or do not want to) modify the non-COBOL driver, you can use the unmodified driver while specifying the LE/VSE RTEREUS run-time option.

(RTEREUS initializes the run-time environment for re-usability when the first COBOL program is invoked.)

Warning: RTEREUS is not recommended for all applications; in some instances, it exhibits undesirable behavior. Before using RTEREUS, you should thoroughly explore the possible side-effects and understand the impact on your application.

APPENDIX1.4 Appendix D. Mixed RES and NORES Applications

VS COBOL II provides the MIXRES run-time option as a feature that aids in migrating from a completely NORES environment to a RES environment. It allows you to mix RES and NORES programs in a single application. The MIXRES run-time option is specified in either the IGZOPD or IGZOPT macro and the resultant options routine is then link-edited with each phase that contains a NORES program.

In addition to the MIXRES run-time option, VS COBOL II provides the assembler macro IGZBRDGV that you can use to convert static calls to dynamic calls. The bridge macro is assembled and the resulting object module is link-edited with the program that contains the static call. This allows you to have an existing program below the 16-megabyte line calling a new program above the line without re-compiling the existing program.

Subtopics:

APPENDIX1.4.1 Implications When Running with LE/VSE

The MIXRES run-time option is not available with LE/VSE. LE/VSE's support of existing applications that specify MIXRES depends on a number of factors, including:

- Whether the application contains a VS COBOL II program, or only DOS/VS COBOL programs
- Whether the program is link-edited with IGZBRDGV
- Whether the program is running in a reusable environment
- Whether the application requires program-specific run-time options or space tuning

Depending on the combination of the above factors, you might need to link-edit the program with LE/VSE.

Programs Requiring Link-Edit with LE/VSE: Any programs compiled with NORES and that specify MIXRES must be link-edited with LE/VSE.

APPENDIX1.4.2 Link-Editing with LE/VSE

Link-editing existing applications with LE/VSE is required in the cases listed above. You might also choose to link-edit other applications with LE/VSE in order to use LE/VSE services. When programs are link-edited with LE/VSE, the application can offer three possibilities: RES behavior, NORES behavior, or unsupported.

Where RES behavior and NORES behavior are defined as:

RES Behavior
Applications that have RES behavior after link-editing with LE/VSE will have access to all LE/VSE services, except LE/VSE callable services.

NORES Behavior
Applications that have NORES behavior after link-editing with LE/VSE will **not** have access to LE/VSE services, but can run under LE/VSE and provide equivalent results as with your current run time.

APPENDIX1.4.3 Behavior after Link-Editing with LE/VSE

The behavior existing programs exhibit after link-editing with LE/VSE depends on whether the application contains at least one VS COBOL II program (including applications with mixed VS COBOL II programs and DOS/VS COBOL programs), or if the application contains DOS/VS COBOL programs only.

For Applications Containing a VS COBOL II Program: Existing applications containing VS COBOL II programs only, or a combination of VS COBOL II and DOS/VS COBOL programs, link-edited with LE/VSE, will provide RES behavior.

For Applications Containing DOS/VS COBOL Programs Only: Existing applications containing DOS/VS COBOL programs only link-edited with LE/VSE, will provide NORES behavior except in the following case:

DOS/VS COBOL with IGZBRDGV:

DOS/VS COBOL applications including IGZBRDGV (from VS COBOL II) or IGZBRDGE (from COBOL/VSE) will exhibit RES behavior after linking with LE/VSE. You must relink all phases in the application with LE/VSE.

Corrective Action: If you need the DOS/VS COBOL application to exhibit RES behavior, you should compile the modules with COBOL/VSE. If this is not possible, you might be able to get it to run by assembling the IGZBRDGV macro and then link-editing the resultant IGZEBRG object module with the above two modules, and then link-editing this new module with LE/VSE. See the *VS COBOL II Application Programming Guide for VSE* for details on using the IGZBRDGV macro.

Note: If you compile one or more programs in an application with COBOL/VSE, you must link-edit the remaining DOS/VS COBOL and/or VS COBOL II NORES programs in the application with LE/VSE in order for those programs to be supported.

APPENDIX1.5 Appendix E. Run-time Option Comparison

This appendix describes all of the LE/VSE run-time options.

Any DOS/VS COBOL or VS COBOL II run-time option equivalents are also shown. Differences in implementation between each LE/VSE option and each DOS/VS COBOL and VS COBOL II option are also briefly described.

For complete descriptions of the LE/VSE run-time options (including their default values, and suboptions where applicable), see the *LE/VSE Programming Guide* and the *LE/VSE Installation and Customization Guide*.

Figure 35. Comparison of LE/VSE, VS COBOL II, and DOS/VS COBOL Run-time Options

LE/VSE	VS COBOL II	DOS/VS COBOL	Usage Notes
ABPERC			Allows the specified abend code to be percolated without going through the LE/VSE condition handling process when TRAP(ON) is in effect. ABPERC is ignored under CICS.
ABTERMENC			Sets the enclave termination behavior for an enclave ending with an unhandled condition of severity 2 or greater. ABTERMENC(ABEND) provides similar abend behavior to VS COBOL II and DOS/VS COBOL.
AIXBLD	AIXBLD	AIXBLD	Indicates that, when indexes are empty at open time, VS COBOL II will invoke access method services to build alternative indexes for VSAM files. AIXBLD is not applicable under CICS. **Note:** For COBOL/VSE and VS COBOL II, AIXBLD is specified by the PARM parameter of the EXEC statement. For DOS/VS COBOL the JCL OPTION SYSPARM is used.
ALL31			Specifies whether or not all application code is running with

			AMODE(31). ALL31(OFF) must be specified if there is a VS COBOL II NORES program, a DOS/VS COBOL program (non-CICS only), or an AMODE(24) non-COBOL program in the application.
ANYHEAP			Controls the amount of heap storage used by the LE/VSE and HLL internal structures.
BELOWHEAP			Controls the amount of heap storage used by the LE/VSE internal structures that must be below the 16-megabyte line.
CBLOPTS			Specifies the format of the character string used during program invocation. CBLOPTS(ON) allows the existing COBOL format to continue working.
CBLPSHPOP			Controls whether CICS PUSH HANDLE and CICS POP HANDLE commands are issued when a COBOL subroutine is called.
CHECK	SSRANGE		Checks index, subscript, and reference modification ranges. CHECK and SSRANGE are synonymous.
COUNTRY			Allows you to choose country-specific default formats. The default supplied by IBM is COUNTRY(US).
DEBUG	DEBUG	DEBUG	Produces debugging information when a WITH DEBUGGING MODE clause is provided in the source program. **Note:** For COBOL/VSE and VS COBOL II, DEBUG is specified by the PARM parameter of the EXEC statement. For DOS/VS COBOL the JCL OPTION SYSPARM is used.
DEPTHCONDLMT			Specifies the extent to which conditions can be nested.
ERRCOUNT			Specifies how many conditions of severity 2, 3, or 4 are allowed to occur before an abend.
FLOW or FLOW(n)			This option is provided for /370 compatibility only.
HEAP			Instructs the LE/VSE storage manager about the size and location of the LE/VSE initial heap allocation.
INTERRUPT			Causes attentions recognized by the host operating system to be recognized by LE/VSE, after the LE/VSE environment has been initialized. INTERRUPT is not applicable under CICS. The option is syntax-checked but has no effect on the run-time behavior of your application.

	LIBKEEP		Causes run-time routines to remain in memory between calls to COBOL RES main programs from non-COBOL programs.
LIBSTACK			Controls the allocation of the enclave's library stack storage below the 16-megabyte line.
	MIXRES		Allows a mixture of RES and NORES compiled programs to reside within a single thread.
MSGFILE			Allows you to specify the filename of the file to which all run-time messages and DISPLAY SYSLST are to be written. MSGFILE is ignored under CICS.
MSGQ			Specifies the number of Message Insert Blocks that will be allocated on a per enclave basis.
NATLANG	LANGUAGE		Specifies the initial national language to be used for the run-time environment. The default supplied by is ENU. NATLANG replaces LANGUAGE in COBOL/VSE
RPTOPTS	SPOUT		Specifies whether a report of the run-time options in use is generated and transmitted to the message file. SPOUT and RPTOPTS are synonymous.
RPTSTG	SPOUT		Produces a report of storage use. Storage information can be used to determine what values are to be specified in the HEAP and STACK options. SPOUT and RPTSTG are synonymous.
RTEREUS	RTEREUS		Initializes the run-time environment for reusability when the first COBOL program is invoked. RTEREUS is not applicable under CICS.
SIMVRD	SIMVRD		Provides simulation of variable length relative organization files. SIMVRD is not applicable under CICS. This option is syntax-checked, but has no effect on the run-time behavior of your application.
STACK			Instructs the LE/VSE storage manager about the size and location of the user stack segment allocation. STACK(,,BELOW) must be specified if there is a VS COBOL II NORES program, a DOS/VS COBOL program (non-CICs only), or an AMODE(24) non-COBOL program in the application.
STORAGE	WSCLEAR		Reserves storage for the short-on-storage condition, and controls the contents of heap storage and DSAs. STORAGE(00) replaces WSCLEAR in COBOL/VSE for a RENT program.

TERMTHDACT			Sets the level of information that is produced when a severity 2 or higher error is not handled.
TEST	TEST		TEST is provided for run-time option compatibility with LE/370. If specified, it is syntax-checked, but it has no effect on the run-time behavior of your application.
TRAP	STAE		Controls whether the LE/VSE Condition Handler will be fully enabled. CEESGL is unaffected by TRAP.
UPSI	UPSI	UPSI	For COBOL/VSE this option is specified by the PARM parameter of the EXEC statement, for DOS/VS COBOL it is specified by the JCL UPSI statement.
VCTRSAVE			Specifies if any language in the application will use the vector facility within the HLL-provided condition handlers. This option is syntax-checked, but it has no effect on the run-time behavior of your application.
XUFLOW			Specifies whether an exponent underflow should cause a program interrupt.

APPENDIX1.6 Appendix F. Compiler Option Comparison

This appendix describes all of the COBOL/VSE compiler options. Figure 36 shows how the COBOL/VSE compiler options compare with those of VS COBOL II and DOS/VS COBOL. For complete descriptions of these options, see the *COBOL/VSE Programming Guide*.

The IBM supplied defaults are indicated in **bold** type.

Figure 36. Compiler Option Comparison

DOS/VS COBOL	**VS COBOL II**	**COBOL/VSE**	**Usage Notes**
(none)	(none)	ADATA	The compiler updates the SYSADAT file
(none)	(none)	**NOADATA**	The compiler does not update the SYSADAT file
ADV	**ADV**	**ADV**	Adds print control byte at beginning of record
NOADV	NOADV	NOADV	Does not add print control byte at beginning of record
(none)	**ALOWCBL**	**ALOWCBL**	(Specified only at installation time.) Allows PROCESS or CBL statements in source program

(none)	NOALOWCBL	NOALOWCBL	Does not allow PROCESS or CBL statements in source program
APOST	APOST	APOST	Specifies apostrophe (') as delimiter for literals
QUOTE	**QUOTE**	**QUOTE**	Specifies quotation mark (") as delimiter for literals
(none)	AWO	AWO	Activates APPLY WRITE-ONLY processing for physical sequential files with VB format
(none)	**NOAWO**	**NOAWO**	Does not activate APPLY WRITE-ONLY processing for physical sequential files with VB format
BUF	BUFSIZE(nnnnn) BUFSIZE(nnnK) **BUFSIZE(4096)**	BUFSIZE(nnnnn) BUFSIZE(nnnK) **BUFSIZE(4096)**	Allocates buffer storage for compiler work files
(none)	CMPR2	CMPR2	Specifies generation of COBOL/VSE code compatible with VS COBOL II Release 2 or other VS COBOL II CMPR2 behavior.
(none)	**NOCMPR2**	**NOCMPR2**	Specifies full use of all COBOL/VSE and later language features.
NOSYNTAX NOCSYNTAX	COMPILE	COMPILE	Requests an unconditional full compilation
SYNTAX CSYNTAX SUPMAP	NOCOMPILE **NOCOMPILE** (W|E|**S**)	NOCOMPILE **NOCOMPILE** (W|E|**S**)	NOCOMPILE/SYNTAX specify unconditional syntax checking. NOCOMPILE (W|E|S)/CSYNTAX/SUPMAP specify conditional syntax checking
(none)	(none)	CURRENCY	Defines default currency symbol
(none)	(none)	**NOCURRENCY**	Indicates that no alternate default currency sign is provided by the CURRENCY option.
(none)	DATA(24)	DATA(24)	Specifies that reentrant program data areas must reside below the 16-megabyte line
(none)	**DATA(31)**	**DATA(31)**	Specifies that reentrant program data areas can reside above or below the 16-megabyte line
(none)	DBCS	DBCS	Tells the compiler to recognize DBCS shift-in and shift-out codes
(none)	**NODBCS**	**NODBCS**	Tells the compiler to treat DBCS shift-in and shift-out codes as ordinary data characters
(none)	DBCSXREF=code	DBCSXREF=code	(Specified only at installation time.) Where **code** sets parameters giving information about the DBCS Ordering Support Program
(none)	**DBCSXREF=NO**	**DBCSXREF=NO**	Specifies that no ordering program is to be used for cross-references to DBCS characters.

DECK (JCL only)	DECK (JCL only)	DECK (JCL only)	Generates object code as 80-character card images and places it in SYSPCH file
NODECK (JCL only)	**NODECK** (JCL only)	**NODECK** (JCL only)	Sends no object code to SYSPCH
(none)	DUMP	DUMP	Specifies that a system dump be produced at end of compilation
(none)	**NODUMP**	**NODUMP**	Specifies that a system dump not be produced at end of compilation
(none)	DYNAM	DYNAM	Loads subprograms dynamically at run time
(none)	**NODYNAM**	**NODYNAM**	Link-edits subprograms statically into the phase
(none)	EXIT (IN-ID,LIB-ID,PRT-ID)	EXIT (IN-ID,LIB-ID,PRT-ID, AD-ID)	Allows the compiler to accept user-supplied modules. (Each *string* is an optional user-supplied input string to the exit module, and each *mod* names a user-supplied exit module.) **IN-ID** Can be either of the following: **INEXIT(['string1',]mod1)** Replaces SYSIN file with *mod1* **NOINEXIT** Does not allow replacement of SYSIN file **LIB-ID** Can be either of the following: **LIBEXIT(['string2',]mod2)** Replaces SYSLIB file with *mod2* **NOLIBEXIT** Does not allow replacement of SYSLIB file **PRT-ID** Can be either of the following: **PRTEXIT(['string3',]mod3)** Replaces SYSPRINT file with *mod3* **NOPRTEXIT** Does not allow replacement of SYSPRINT file **AD-ID** For **COBOL/VSE only !** This read-only exit will 'see' all the ADATA records. Any modifications to the ADATA records will be ignored. It can be either of the following: **ADEXIT(['string4',]mod4)** Specifies that a user-supplied associated-data (SYSADAT) exit is to be used for the compilation. *mod4* is the name of the phase for that exit. **NOADEXIT** Specifies that a user-supplied

			associated-data (SYSADAT) exit is not to be used for the compilation.
(none)	**NOEXIT**	**NOEXIT**	Tells the compiler not to accept user-supplied modules.
(none)	FASTSRT	FASTSRT	Specifies fast sorting by the IBM DFSORT* licensed program
(none)	**NOFASTSRT**	**NOFASTSRT**	Specifies that COBOL/VSE is to do sort/merge input/output
FLAGW FLAGE	**FLAG(I)** FLAG(W) FLAG(E) FLAG(S) FLAG(U) FLAG(I|W|E|S|U,I|W|E|S|U)	**FLAG(I)** FLAG(W) FLAG(E) FLAG(S) FLAG(U) FLAG(I|W|E|S|U,I|W|E|S|U)	Specifies that syntax messages be produced, at level indicated
(none)	NOFLAG	NOFLAG	Specifies that syntax messages not be produced
(none)	FLAGMIG	FLAGMIG	Specifies COBOL/VSE NOCMPR2 flagging for possible semantic changes from VS COBOL II Release 2 or other programs with CMPR2 behavior.
(none)	**NOFLAGMIG**	**NOFLAGMIG**	No COBOL/VSE flagging for possible semantic changes
(none)	FLAGSAA	FLAGSAA	Specifies COBOL/VSE Systems Application Architecture flagging
(none)	**NOFLAGSAA**	**NOFLAGSAA**	No COBOL/VSE Systems Application Architecture flagging
(none)	FLAGSTD	FLAGSTD	Specifies COBOL 85 FIPS and ANS flagging
(none)	**NOFLAGSTD**	**NOFLAGSTD**	No COBOL 85 FIPS and ANS flagging
(none)	**LANGUAGE(EN)** LANGUAGE(AAa..a)	**LANGUAGE(UE)** LANGUAGE(AAa..a)	**AAa..a**, specifies language in which compiler messages are issued: UE or UENGLISH Uppercase English EN or ENGLISH Mixed-case English JA, JP, or JAPANESE Japanese, using the KANJI character set
LIB	LIB	LIB	Specifies that this program uses the COPY library
NOLIB	**NOLIB**	**NOLIB**	Specifies that this program does not use the COPY library
LINECNT=nn (Install only)	**LINECOUNT(60)** LINECOUNT(nnn)	**LINECOUNT(60)** LINECOUNT(nnn)	Specifies the number of lines per page on the output listing
LISTX (JCL only)	LIST	LIST	Produces listing of assembler language expansion of source code
NOLISTX (JCL only)	**NOLIST**	**NOLIST**	Does not produce listing of assembler language expansion of source code

SYM (JCL only)	MAP	MAP	Produces listing of Data Division and implicitly declared items
NOSYM (JCL only)	**NOMAP**	**NOMAP**	Does not produce listing of Data Division and implicitly declared items
CATALR	NAME (ALIAS|NOALIAS)	NAME (ALIAS|NOALIAS)	Produces VSE/Librarian CATALOG statement and/or linkage editor phase statement for each object module created. DOS/VS COBOL only produces a CATALR statement at the beginning of the object deck.
NOCATLR	**NONAME**	**NONAME**	No VSE/Librarian CATALOG statement or linkage editor PHASE statement produced.
(none)	NUMBER	NUMBER	Prints line numbers in error messages and listings
(none)	**NONUMBER**	**NONUMBER**	Does not print line numbers in error messages and listings
(none)	**NUMCLS(PRIM)** NUMCLS(ALT)	**NUMCLS(PRIM)** NUMCLS(ALT)	(Specified only at installation time.) Determines, together with the NUMPROC option, valid sign configurations for numeric items in the NUMERIC class test. See *COBOL/VSE Programming Guide* for more information.
(none)	NUMPROC(PFD) **NUMPROC(NOPFD)** NUMPROC(MIG)	NUMPROC(PFD) **NUMPROC(NOPFD)** NUMPROC(MIG)	Handles COBOL/VSE packed/zoned decimal signs as follows: NUMPROC(PFD) Decimal fields assumed to have standard System/370 signs NUMPROC(NOPFD) The compiler does any necessary sign conversion and repair NUMPROC(MIG) COBOL/VSE processes sign conversion in a manner very similar to DOS/VS COBOL
LINK (JCL only)	OBJECT	OBJECT	Writes object code to SYSLNK for input to linkage editor
NOLINK **(JCL only)**	**NOOBJECT**	**NOOBJECT**	Does not write object code to SYSLNK for input to linkage editor
CLIST	OFFSET	OFFSET	Produces condensed Procedure Division listing plus tables and program statistics
NOCLIST	**NOOFFSET**	**NOOFFSET**	Does not produce condensed Procedure Division listing nor tables and program statistics
OPTIMIZE	OPTIMIZE	OPTIMIZE	Optimizes object program. (COBOL/VSE OPTIMIZE provides significantly more optimization than DOS/VS COBOL)
NOOPTIMIZE	**NOOPTIMIZE**	**NOOPTIMIZE**	Does not optimize object program.
(none)	**(none)**	**OUTDD(SYSLST)** OUTDD(filename)	Routes DISPLAY output to SYSLST or to a specified file

(none)	RENT	RENT	Specifies reentrant code in object program
(none)	**NORENT**	**NORENT**	Specifies nonreentrant code in object program
(none)	(none)	RMODE (**AUTO**|24|ANY)	Allows NORENT programs to have RMODE(ANY)
SEQ	**SEQUENCE**	**SEQUENCE**	Checks ascending sequencing of source statement line numbers
NOSEQ	NOSEQUENCE	NOSEQUENCE	Does not check ascending sequencing of source statement line numbers
SIZE=nnnnn (JCL only)	**SIZE(MAX)** SIZE(nnnnnn) SIZE(nnnK)	**SIZE(MAX)** SIZE(nnnnnn) SIZE(nnnK)	Specifies virtual storage to be used for compilation
LIST (JCL only)	**SOURCE**	**SOURCE**	Produces a listing of the source program and embedded messages
NOLIST (JCL only)	NOSOURCE	NOSOURCE	Does not produce a listing of the source program and messages
SPACE1 SPACE2 SPACE3	**SPACE(1)** SPACE(2) SPACE(3)	**SPACE(1)** SPACE(2) SPACE(3)	Produces a single, double, or triple spaced listing
(none)	SSRANGE	SSRANGE	At run time, checks validity of subscript, index, and reference modification references
(none)	**NOSSRANGE**	**NOSSRANGE**	At run time, suppresses checking for validity of subscript, index, and reference modification references
(none)	TERMINAL	TERMINAL	Sends progress messages to SYSLOG
(none)	**NOTERMINAL**	**NOTERMINAL**	Does not send progress messages to SYSLOG
(none)	TEST	TEST (other than NO)	Symbol table information is produced to allow symbolic variables to be included in a formatted dump produced by LE/VSE.
(none)	**NOTEST**	**NOTEST**	Symbol table information is not produced, and symbolic variables are not included in a formatted dump produced by LE/VSE.
TRUNC NOTRUNC	**TRUNC(STD)** TRUNC(OPT) TRUNC(BIN)	**TRUNC(STD)** TRUNC(OPT) TRUNC(BIN)	Shortens intermediate results, as follows: TRUNC(STD) Shortens numeric fields according to PICTURE specification of the binary receiving field TRUNC(OPT) Shortens numeric fields in the most optimal way TRUNC(BIN) Shortens binary fields based on the storage they occupy See *COBOL/VSE Programming Guide* for a complete description

VERBREF VERBSUM	VBREF	VBREF	Produces cross-reference listing of all verb types used in program
NOVERBREF **NOVERBSUM**	**NOVBREF**	**NOVBREF**	Does not produce cross-reference listing of all verb types used in program
(none)	WORD(table-name)	WORD(table-name)	Tells the compiler to use an installation-specified reserved word table
(none)	**NOWORD**	**NOWORD**	Tells the compiler to use the default reserved word table
SXREF	XREF XREF(FULL) XREF(SHORT)	XREF XREF(FULL) XREF(SHORT)	Produces sorted cross-reference listing of data names and procedure names used in program
NOSXREF	**NOXREF**	**NOXREF**	Does not produce sorted cross-reference listing of data names and procedure names used in program
ZWB	**ZWB**	**ZWB**	Removes sign from a signed numeric DISPLAY field when comparing it with an alphanumeric field
NOZWB	NOZWB	NOZWB	Does not remove sign from a signed numeric DISPLAY field when comparing it with an alphanumeric field

APPENDIX1.7 Appendix G. Compiler Limit Comparison

The following table lists the compiler limits for COBOL/VSE, VS COBOL II, and DOS/VS COBOL programs executing under VSE/ESA. Other operating systems might impose further limits.

The numbers are **guidelines** to the limits.

- A limit stated in megabytes (M) should be interpreted as: x megabytes minus 1 byte.
- A limit stated in kilobytes (K) should be interpreted as: x kilobytes minus 1 byte.
- A limit stated in gigabytes (G) should be interpreted as: x gigabytes minus 1 byte.
- B stands for bytes.
- N/A stands for not applicable.
- N/L stands for no limit.
- Footnotes are at the end of the table.

Language Element	COBOL/VSE	VS COBOL II	DOS/VS COBOL
Size of program	999,999 lines	999,999 lines	1M lines
Number of literals	4,194,303B(1)	4,194,303B(1)	16K
Total length of literals	4,194,303B(1)	4,194,303B(1)	32K after OPT
Reserved Word Table entries	1536	1536	N/A
COPY REPLACING ... BY ... (items per COPY statement)	N/A	N/A	150
Number of COPY libraries	N/A	N/A	N/L
Identification Division			
Environment Division			
Configuration Section			
SPECIAL-NAMES paragraph			
function-name IS	18	18	18
UPSI-n ... (switches)	0-7	0-7	0-7
alphabet-name IS ...	N/A	N/A	N/L
literal THRU/ALSO ...	256	256	256
Input-Output Section			
FILE-CONTROL paragraph			
SELECT file-name ...	65,535	65,535	64K
ASSIGN system-name ...	N/A(2)	N/A(3)	N/L(2)
ALTERNATE RECORD KEY data-name ...	253	253	253
RECORD KEY length	N/A(3)	N/A(3)	255
RESERVE integer (buffers)	255(4)	255(4)	255(4)
I-O-CONTROL paragraph			
RERUN ON system-name ...	32,767	32,767	32K
integer RECORDS	16,777,215	16,777,215	16M
SAME RECORD AREA	255	255	255
FOR file-name ...	255	255	255
SAME SORT/MERGE AREA	N/A(2)	N/A(2)	N/L(2)
MULTIPLE FILE ... file-name(8)	N/A(2)	N/A(2)	N/L(2)
Data Division			
File Section			
FD file-name ...	65,535	65,535	64K
LABEL data-name ... (if no optional clauses)	255	255	185
Label record length	80B	80B	80B
DATA RECORD dnm ...	N/A(2)	N/A(2)	N/L(2)
BLOCK CONTAINS integer	1,048,575(5)	1,048,575(5)	32760
RECORD CONTAINS integer	1,048,575(5)	1,048,575(5)	32K
Item length	1,048,575(5)	1,048,575(5)	32K
SD file-name ...	65,535	65,535	64K
DATA RECORD dnm ...	N/A(2)	N/A(2)	N/L(2)
Sort record length	32,751B	32,751B	32K-16B
Working-Storage Section (items w/o the EXTERNAL attribute)	134,217,727B	134,217,727B	1M

Working-Storage Section (items with the EXTERNAL attribute)	134,217,727B	134,217,727B	1M
77 data-names	16,777,215	16,777,215	1M
01-49 data-names	16,777,215	16,777,215	1M
88 condition-name ...	N/A	N/A	N/L
VALUE literal ...	N/A	N/A	N/L
66 RENAMES ...	N/A	N/A	N/L
PICTURE character string	30	30	30
Numeric item digit positions	18	18	18
Num-edit character positions	249	249	127
PICTURE replication ( )	16,777,215	16,777,215	99999
PIC repl (editing)	32,767	32,767	99999
DBCS Picture Reactivation	8,388,607		
Group item size:			
Linkage section	16,777,215	16,777,215	131K
Other sections	16,777,215	16,777,215	32K
Elementary item size	16,777,215	16,777,215	32K
VALUE initialization (Total length of value literals)	16,777,215	16,777,215	64K
OCCURS integer	16,777,215	16,777,215	32K
Total number of ODOs	4,194,303(1)	4,194,303(1)	64K(1)
Levels of ODO	N/A	N/A	3
Table size	16,777,215	16,777,215	32K
Table element	8,388,607	8,388,607	32K
ASCENDING/DESCENDING KEY ... (per OCCURS clause)	12	12	12
Total length	256B	256B	256B
INDEXED BY ... (index names)	12	12	12
Total num of indexes (index names)	65,535	65,535	64K
Size of relative index	32,765	32,765	32K
Linkage Section	134,217,727	134,217,727	1M
Total 01 + 77 (data items)	N/A	N/A	255
Procedure Division			
Procedure + constant area	4,194,303(1)	4,194,303(1)	1M+32K
USING identifier ...	32,767	32,767	N/L
Procedure-names	1,048,575(1)	1,048,575(1)	64K(1)
Verbs per line (FDUMP/TEST)	7	7	7
Subscripted data-names per verb	32,767	32,767	511
ADD identifier ...	N/A	N/A	N/L
ALTER pn1 TO pn2 ...	4,194,303(1)	4,194,303(1)	64K
CALL ... BY CONTENT id	2,147,483,647	2,147,483,647	N/A
CALL literal ...	4,194,303(1)	4,194,303(1)	N/L
USING id/lit ...	16,380	16,380	N/L
Active programs in run unit	32,767	32,767	32K
RES/DYN number of names called	N/A	N/A	64K
CANCEL id/lit ...	N/A	N/A	N/L
CLOSE file-name ...	N/A	N/A	N/L
COMPUTE identifier ...	N/A	N/A	N/L
DISPLAY id/lit ...	N/A	N/A	N/L
DIVIDE identifier ...	N/A	N/A	N/L
ENTRY USING id/lit ...	N/A	N/A	N/L
EVALUATE ... subjects	64	64	N/A
EVALUATE ... WHEN clauses	256	256	N/A
GO pn ... DEPENDING	255	255	2031
INSPECT TALLYING/REPLACING	N/A	N/A	15
MERGE file-name			
ASCENDING/DESCENDING KEY ...	N/A	N/A	12
Total key length	4,092B(6)	4092B(6)	256
USING file-name ...	16(7)	16(7)	16(7)
MOVE id/lit TO id ...	N/A	N/A	N/L
MULTIPLY identifier ...	N/A	N/A	N/L
OPEN file-name	N/A	N/A	N/L
PERFORM	4,194,303	4,194,303	64K
SEARCH ... WHEN ...	N/A	N/A	N/L
SEARCH ALL ... WHEN ...	12	12	12
SET index/id ... TO	N/A	N/A	N/L
SET index ... UP/DOWN	N/A	N/A	N/L

SORT file-name			
ASCENDING/DESCENDING KEY	N/A	N/A	12
Total key length	4,092B(6)	4092B(6)	256
USING file-name ...	16(7)	16(7)	16(7)
STRING identifier ...	N/A	N/A	N/L
DELIMITED id/lit ...	N/A	N/A	N/L
UNSTRING id			
DELIMITED id/lit OR id/lit ...	255	255	15
INTO id/lit ...	N/A	N/A	N/L
USE ... ON file-name ...	N/A	N/A	N/L

For notes regarding this table this the following page.

Notes:

(1) Items included in 4,194,303 byte limit for procedure plus constant area
(2) Treated as comment; there is no limit
(3) No compiler limit, but VSAM limits it to 255 bytes
(4) SAM limit is 2
(5) Compiler limit shown, but SAM limits it to 32,767 bytes
(6) The DFSORT/VSE limit is 3,072
(7) The DFSORT/VSE limit is 9
(8) The MULTIPLE FILE TAPE phrase is ignored in COBOL/VSE

APPENDIX1.8 Appendix H. COBOL Reserved Word Comparison

This appendix contains a table showing differences between DOS/VS COBOL, VS COBOL II, and COBOL/VSE reserved words.

This list identifies all reserved words in COBOL/VSE, VS COBOL II, and DOS/VS COBOL.

New COBOL/VSE reserved words are highlighted in **boldface** type.

Key:

X	The word is reserved in the product.
- (hyphen)	The word is *not* reserved in the product. (This includes obsolete reserved words that are no longer flagged.)
CDW	The word is a COBOL/VSE compiler directing statement. If used as a user-defined word, it is flagged with a severe message.
COD	The word is a CODASYL reserved word not used by this compiler. If used, it is recognized as a user-defined word and flagged with an informational message.
SYS	The word is a word with specific meaning to the operating system. It can be used only in specific contexts within the program.
UNS	The word is a COBOL 1985 Standard reserved word for a feature not supported by this compiler. If used in a program, it is recognized as a reserved word and flagged with a severe message.

Figure 37. Reserved Words

Reserved Word	/VSE	VS II	DOS/VS
ACCEPT	X	X	X
ACCESS	X	X	X
ACTUAL	-	-	X
ADD	X	X	X
ADDRESS	X	X	X
ADVANCING	X	X	X
AFTER	X	X	X
ALL	X	X	X
ALPHABET	X	X	-
ALPHABETIC	X	X	X
ALPHABETIC-LOWER	X	X	-
ALPHABETIC-UPPER	X	X	-
ALPHANUMERIC	X	X	-
ALPHANUMERIC-EDITED	X	X	-
ALSO	X	X	X
ALTER	X	X	X
ALTERNATE	X	X	X
AND	X	X	X
ANY	X	X	-
APPLY	X	X	X
ARE	X	X	X
AREA	X	X	X
AREAS	X	X	X
ARITHMETIC	COD	COD	-
ASCENDING	X	X	X
ASSIGN	X	X	X
AT	X	X	X
AUTHOR	X	X	X
B-AND	COD	COD	-
B-EXOR	COD	COD	-
B-LESS	COD	COD	-
B-NOT	COD	COD	-
B-OR	COD	COD	-

BASIS	CDW	CDW	X
BEFORE	X	X	X
BEGINNING	X	X	X
BINARY	X	X	-
BIT	COD	COD	-
BITS	COD	COD	-
BLANK	X	X	X
BLOCK	X	X	X
BOOLEAN	COD	COD	-
BOTTOM	X	X	X
BY	X	X	X
CALL	X	X	X
CANCEL	X	X	X
CBL	CDW	CDW	X
CD	UNS	UNS	X
CF	UNS	UNS	X
CH	UNS	UNS	X
CHANGED	-	-	X
CHARACTER	X	X	X
CHARACTERS	X	X	X
CLASS	X	X	-
CLOCK-UNITS	UNS	UNS	-
CLOSE	X	X	X
COBOL	X	X	-
CODE	X	X	X
CODE-SET	X	X	X
COLLATING	X	X	X
COLUMN	UNS	UNS	X
COM-REG	X	X	-
COMMA	X	X	X
COMMIT	COD	COD	-
COMMON	X	X	-
COMMUNICATION	UNS	UNS	X
COMP	X	X	X
COMP-1	X	X	X
COMP-2	X	X	X

COMP-3	X	X	X
COMP-4	X	X	X
COMP-5	COD	COD	-
COMP-6	COD	COD	-
COMP-7	COD	COD	-
COMP-8	COD	COD	-
COMP-9	COD	COD	-
COMPUTATIONAL	X	X	X
COMPUTATIONAL-1	X	X	X
COMPUTATIONAL-2	X	X	X
COMPUTATIONAL-3	X	X	X
COMPUTATIONAL-4	X	X	X
COMPUTATIONAL-5	COD	COD	-
COMPUTATIONAL-6	COD	COD	-
COMPUTATIONAL-7	COD	COD	-
COMPUTATIONAL-8	COD	COD	-
COMPUTATIONAL-9	COD	COD	-
COMPUTE	X	X	X
CONFIGURATION	X	X	X
CONNECT	COD	COD	-
CONSOLE	SYS	SYS	X
CONTAINED	COD	COD	-
CONTAINS	X	X	X
CONTENT	X	X	-
CONTINUE	X	X	-
CONTROL	UNS	UNS	X
CONTROLS	UNS	UNS	X
CONVERTING	X	X	-
COPY	X	X	X
CORR-INDEX	-	-	X
CORR	X	X	X
CORRESPONDING	X	X	X
COUNT	X	X	X
CSP	SYS	SYS	X
CURRENCY	X	X	X
CURRENT	COD	COD	-

CURRENT-DATE	-	-	X
C01	SYS	SYS	X
C02	SYS	SYS	X
C03	SYS	SYS	X
C04	SYS	SYS	X
C05	SYS	SYS	X
C06	SYS	SYS	X
C07	SYS	SYS	X
C08	SYS	SYS	X
C09	SYS	SYS	X
C10	SYS	SYS	X
C11	SYS	SYS	X
C12	SYS	SYS	X
DATA	X	X	X
DATE	X	X	X
DATE-COMPILED	X	X	X
DATE-WRITTEN	X	X	X
DAY	X	X	X
DAY-OF-WEEK	X	X	-
DB	COD	COD	-
DB-ACCESS-CONTROL-KEY	COD	COD	-
DB-DATA-NAME	COD	COD	-
DB-EXCEPTION	COD	COD	-
DB-RECORD-NAME	COD	COD	-
DB-SET-NAME	COD	COD	-
DB-STATUS	COD	COD	-
DBCS	X	X	-
DE	UNS	UNS	X
DEBUG	-	-	X
DEBUG-CONTENTS	X	X	X
DEBUG-ITEM	X	X	X
DEBUG-LINE	X	X	X
DEBUG-NAME	X	X	X
DEBUG-SUB-1	X	X	X
DEBUG-SUB-2	X	X	X
DEBUG-SUB-3	X	X	X

DEBUGGING	X	X	X
DECIMAL-POINT	X	X	X
DECLARATIVES	X	X	X
DEFAULT	COD	COD	-
DELETE	X	X	X
DELIMITED	X	X	X
DELIMITER	X	X	X
DEPENDING	X	X	X
DESCENDING	X	X	X
DESTINATION	UNS	UNS	X
DETAIL	UNS	UNS	X
DISABLE	UNS	UNS	X
DISCONNECT	COD	COD	-
DISP	-	-	X
DISPLAY	X	X	X
DISPLAY-ST	-	-	X
DISPLAY-1	X	X	-
DISPLAY-2	COD	COD	-
DISPLAY-3	COD	COD	-
DISPLAY-4	COD	COD	-
DISPLAY-5	COD	COD	-
DISPLAY-6	COD	COD	-
DISPLAY-7	COD	COD	-
DISPLAY-8	COD	COD	-
DISPLAY-9	COD	COD	-
DIVIDE	X	X	X
DIVISION	X	X	X
DOWN	X	X	X
DUPLICATE	COD	COD	-
DUPLICATES	X	X	X
DYNAMIC	X	X	X
EGCS	X	X	-
EGI	UNS	UNS	X
EJECT	CDW	CDW	X
ELSE	X	X	X
EMI	UNS	UNS	X

EMPTY	COD	COD	-
ENABLE	UNS	UNS	X
END	X	X	X
END-ADD	X	X	-
END-CALL	X	X	-
END-COMPUTE	X	X	-
END-DELETE	X	X	-
END-DISABLE	COD	COD	-
END-DIVIDE	X	X	-
END-ENABLE	COD	COD	-
END-EVALUATE	X	X	-
END-IF	X	X	-
END-MULTIPLY	X	X	-
END-OF-PAGE	X	X	X
END-PERFORM	X	X	-
END-READ	X	X	-
END-RECEIVE	UNS	UNS	-
END-RETURN	X	X	-
END-REWRITE	X	X	-
END-SEARCH	X	X	-
END-SEND	COD	COD	-
END-START	X	X	-
END-STRING	X	X	-
END-SUBTRACT	X	X	-
END-TRANSCEIVE	COD	COD	-
END-UNSTRING	X	X	-
END-WRITE	X	X	-
ENDING	X	X	X
ENTER	X	X	X
ENTRY	X	X	X
ENVIRONMENT	X	X	X
EOP	X	X	X
EQUAL	X	X	X
EQUALS	COD	COD	-
ERASE	COD	COD	-
ERROR	X	X	X

ESI	UNS	UNS	X
EVALUATE	X	X	-
EVERY	X	X	X
EXACT	COD	COD	-
EXAMINE	-	-	X
EXCEEDS	COD	COD	-
EXCEPTION	X	X	X
EXCLUSIVE	COD	COD	-
EXHIBIT	-	-	X
EXIT	X	X	X
EXTEND	X	X	X
EXTERNAL	X	X	-
FALSE	X	X	-
FD	X	X	X
FETCH	COD	COD	-
FILE	X	X	X
FILE-CONTROL	X	X	X
FILE-LIMIT	-	-	X
FILE-LIMITS	-	-	X
FILLER	X	X	X
FINAL	UNS	UNS	X
FIND	COD	COD	-
FINISH	COD	COD	-
FIRST	X	X	X
FOOTING	X	X	X
FOR	X	X	X
FORMAT	COD	COD	-
FREE	COD	COD	-
FROM	X	X	X
FUNCTION	X	COD	-
GENERATE	UNS	UNS	X
GET	COD	COD	-
GIVING	X	X	X
GLOBAL	X	X	-
GO	X	X	X
GOBACK	X	X	X

GREATER	X	X	X
GROUP	UNS	UNS	X
HEADING	UNS	UNS	X
HIGH-VALUE	X	X	X
HIGH-VALUES	X	X	X
I-O	X	X	X
I-O-CONTROL	X	X	X
ID	X	X	X
IDENTIFICATION	X	X	X
IF	X	X	X
IN	X	X	X
INDEX	X	X	X
INDEX-1	COD	COD	-
INDEX-2	COD	COD	-
INDEX-3	COD	COD	-
INDEX-4	COD	COD	-
INDEX-5	COD	COD	-
INDEX-6	COD	COD	-
INDEX-7	COD	COD	-
INDEX-8	COD	COD	-
INDEX-9	COD	COD	-
INDEXED	X	X	X
INDICATE	UNS	UNS	X
INITIAL	X	X	X
INITIALIZE	X	X	X
INITIATE	UNS	UNS	X
INPUT	X	X	X
INPUT-OUTPUT	X	X	X
INSERT	CDW	CDW	X
INSPECT	X	X	X
INSTALLATION	X	X	X
INTO	X	X	X
INVALID	X	X	X
IS	X	X	X
JUST	X	X	X
JUSTIFIED	X	X	X

KANJI	X	X	-
KEEP	COD	COD	-
KEY	X	X	X
LABEL	X	X	X
LAST	UNS	UNS	X
LD	COD	COD	-
LEADING	X	X	X
LEAVE	-	-	X
LEFT	X	X	X
LENGTH	X	X	X
LESS	X	X	X
LIMIT	UNS	UNS	X
LIMITS	UNS	UNS	X
LINAGE	X	X	X
LINAGE-COUNTER	X	X	X
LINE	X	X	X
LINE-COUNTER	UNS	UNS	X
LINES	X	X	X
LINKAGE	X	X	X
LOCALLY	COD	COD	-
LOCK	X	X	X
LOW-VALUE	X	X	X
LOW-VALUES	X	X	X
MEMBER	COD	COD	-
MEMORY	X	X	X
MERGE	X	X	X
MESSAGE	UNS	UNS	X
MODE	X	X	X
MODIFY	COD	COD	-
MODULES	X	X	X
MORE-LABELS	X	X	X
MOVE	X	X	X
MULTIPLE	X	X	X
MULTIPLY	X	X	X
NAMED	-	-	X
NATIVE	X	X	X

NEGATIVE	X	X	X
NEXT	X	X	X
NO	X	X	X
NOMINAL	-	-	X
NONE	COD	COD	-
NOT	X	X	X
NOTE	-	-	X
NULL	X	X	-
NULLS	X	X	-
NUMBER	UNS	UNS	X
NUMERIC	X	X	X
NUMERIC-EDITED	X	X	-
OBJECT-COMPUTER	X	X	X
OCCURS	X	X	X
OF	X	X	X
OFF	X	X	X
OMITTED	X	X	X
ON	X	X	X
ONLY	COD	COD	-
OPEN	X	X	X
OPTIONAL	X	X	X
OR	X	X	X
ORDER	X	X	-
ORGANIZATION	X	X	X
OTHER	X	X	-
OTHERWISE	-	-	X
OUTPUT	X	X	X
OVERFLOW	X	X	X
OWNER	COD	COD	-
PACKED-DECIMAL	X	X	-
PADDING	X	X	-
PAGE	X	X	X
PAGE-COUNTER	UNS	UNS	X
PARAGRAPH	COD	COD	-
PASSWORD	X	X	X
PERFORM	X	X	X

PF	UNS	UNS	X
PH	UNS	UNS	X
PIC	X	X	X
PICTURE	X	X	X
PLUS	UNS	UNS	X
POINTER	X	X	X
POSITION	X	X	X
POSITIONING	-	-	X
POSITIVE	X	X	X
PRESENT	COD	COD	-
PRINT-SWITCH	-	-	X
PRINTING	UNS	UNS	-
PRIOR	COD	COD	-
PROCEDURE	X	X	X
PROCEDURE-POINTER	X	-	-
PROCEDURES	X	X	X
PROCEED	X	X	X
PROCESSING	X	X	X
PROGRAM	X	X	X
PROGRAM-ID	X	X	X
PROTECTED	COD	COD	-
PURGE	UNS	UNS	-
QUEUE	UNS	UNS	X
QUOTE	X	X	X
QUOTES	X	X	X
RANDOM	X	X	X
RD	UNS	UNS	X
READ	X	X	X
READY	X	X	X
REALM	COD	COD	-
RECEIVE	UNS	UNS	X
RECONNECT	COD	COD	-
RECORD	X	X	X
RECORD-NAME	COD	COD	-
RECORD-OVERFLOW	-	-	X
RECORDING	X	X	X

RECORDS	X	X	X
REDEFINES	X	X	X
REEL	X	X	X
REFERENCE	X	X	-
REFERENCES	X	X	X
RELATION	COD	COD	-
RELATIVE	X	X	X
RELEASE	X	X	X
RELOAD	X	X	X
REMAINDER	X	X	X
REMARKS	-	-	X
REMOVAL	X	X	X
RENAMES	X	X	X
REORG-CRITERIA	-	-	X
REPEATED	COD	COD	-
REPLACE	X	X	-
REPLACING	X	X	X
REPORT	UNS	UNS	X
REPORTING	UNS	UNS	X
REPORTS	UNS	UNS	X
REREAD	-	-	X
RERUN	X	X	X
RESERVE	X	X	X
RESET	-	-	X
RETAINING	COD	COD	-
RETRIEVAL	COD	COD	-
RETURN	X	X	X
RETURN-CODE	X	X	X
REVERSED	X	X	X
REWIND	X	X	X
REWRITE	X	X	X
RF	UNS	UNS	X
RH	UNS	UNS	X
RIGHT	X	X	X
ROLLBACK	COD	COD	-
ROUNDED	X	X	X

RUN	X	X	X
SAME	X	X	X
SD	X	X	X
SEARCH	X	X	X
SECTION	X	X	X
SECURITY	X	X	X
SEEK	-	-	X
SEGMENT	UNS	UNS	X
SEGMENT-LIMIT	X	X	X
SELECT	X	X	X
SELECTIVE	-	-	X
SEND	UNS	UNS	X
SENTENCE	X	X	X
SEPARATE	X	X	X
SEQUENCE	X	X	X
SEQUENTIAL	X	X	X
SERVICE	X	X	X
SESSION-ID	COD	COD	-
SET	X	X	X
SHARED	COD	COD	-
SHIFT-IN	X	X	-
SHIFT-OUT	X	X	-
SIGN	X	X	X
SIZE	X	X	X
SKIP-1	-	-	X
SKIP-2	-	-	X
SKIP-3	-	-	X
SKIP1	CDW	CDW	-
SKIP2	CDW	CDW	-
SKIP3	CDW	CDW	-
SORT	X	X	X
SORT-CONTROL	X	X	-
SORT-CORE-SIZE	X	X	X
SORT-FILE-SIZE	X	X	X
SORT-MERGE	X	X	X
SORT-MESSAGE	X	X	X

SORT-MODE-SIZE	X	X	X
SORT-RETURN	X	X	X
SOURCE	UNS	UNS	X
SOURCE-COMPUTER	X	X	X
SPACE	X	X	X
SPACES	X	X	X
SPECIAL-NAMES	X	X	X
STANDARD	X	X	X
STANDARD-1	X	X	X
STANDARD-2	X	X	-
STANDARD-3	COD	COD	-
STANDARD-4	COD	COD	-
START	X	X	X
STATUS	X	X	X
STOP	X	X	X
STORE	COD	COD	X
STRING	X	X	X
SUB-QUEUE-1	UNS	UNS	X
SUB-QUEUE-2	UNS	UNS	X
SUB-QUEUE-3	UNS	UNS	X
SUB-SCHEMA	COD	COD	-
SUBTRACT	X	X	X
SUM	UNS	UNS	X
SUPPRESS	X	X	X
SYMBOLIC	X	X	X
SYNC	X	X	X
SYNCHRONIZED	X	X	X
SYSIN	SYS	SYS	X
SYSIPT	SYS	SYS	-
SYSLIST	SYS	SYS	X
SYSLST	SYS	SYS	-
SYSOUT	SYS	SYS	X
SYSPCH	SYS	SYS	-
SYSPUNCH	SYS	SYS	X
S01	SYS	SYS	X
S02	SYS	SYS	X

S03	SYS	SYS	-
S04	SYS	SYS	-
S05	SYS	SYS	-
TABLE	UNS	UNS	X
TALLY	X	X	X
TALLYING	X	X	X
TAPE	X	X	X
TENANT	COD	COD	-
TERMINAL	UNS	UNS	X
TERMINATE	UNS	UNS	X
TEST	X	X	-
TEXT	UNS	UNS	X
THAN	X	X	X
THEN	X	X	X
THROUGH	X	X	X
THRU	X	X	X
TIME	X	X	X
TIME-OF-DAY	-	-	X
TIMES	X	X	X
TITLE	CDW	CDW	-
TO	X	X	X
TOP	X	X	X
TOTALED	-	-	X
TOTALING	-	-	X
TRACE	X	X	X
TRACK-AREA	-	-	X
TRACK-LIMIT	-	-	X
TRACKS	-	-	X
TRAILING	X	X	X
TRANSCEIVE	COD	COD	-
TRANSFORM	-	-	X
TRUE	X	X	-
TYPE	UNS	UNS	X
UNEQUAL	COD	COD	-
UNIT	X	X	X
UNSTRING	X	X	X

UNTIL	X	X	X
UP	X	X	X
UPDATE	COD	COD	-
UPON	X	X	X
UPSI-0	SYS	SYS	X
UPSI-1	SYS	SYS	X
UPSI-2	SYS	SYS	X
UPSI-3	SYS	SYS	X
UPSI-4	SYS	SYS	X
UPSI-5	SYS	SYS	X
UPSI-6	SYS	SYS	X
UPSI-7	SYS	SYS	X
USAGE	X	X	X
USAGE-MODE	COD	COD	-
USE	X	X	X
USING	X	X	X
VALID	COD	COD	-
VALIDATE	COD	COD	-
VALUE	X	X	X
VALUES	X	X	X
VARYING	X	X	X
WAIT	COD	COD	-
WHEN	X	X	X
WHEN-COMPILED	X	X	X
WITH	X	X	X
WITHIN	COD	COD	-
WORDS	X	X	X
WORKING-STORAGE	X	X	X
WRITE	X	X	X
WRITE-ONLY	X	X	X
ZERO	X	X	X
ZEROES	X	X	X
ZEROS	X	X	X
<	X	X	X
<=	X	X	-
+	X	X	X

*	X	X	X
**	X	X	X
-	X	X	X
/	X	X	X
>	X	X	X
>=	X	X	-
=	X	X	X

APPENDIX1.9 Appendix I. Industry Standards

COBOL/VSE supports the following industry standards in the VSE/ESA environment, as understood and interpreted by IBM as of September 1989:

1. ISO 1989:1985, Programming Languages - COBOL

   ISO 1989/Amendment 1, Programming Languages - COBOL - Amendment 1: Intrinsic function module.

   ISO 1989:1985 is identical to X3.23-1985, American National Standard for Information Systems - Programming Language - COBOL.
   ISO 1989/Amendment 1 is identical to X3.23a-1989, American National Standard for Information Systems - Programming Language - Intrinsic Function Module for COBOL.

   For supported modules, see item 2 below.

2. X3.23-1985, American National Standard for Information Systems - Programming Language - COBOL.

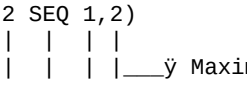
   X3.23a-1989, American National Standard for Information Systems - Programming Language - Intrinsic Function Module for COBOL.

   COBOL/VSE supports all required modules at the intermediate level. It also supports all required modules at the high level with the exception of the following language features:

   - EXTEND phrase of the OPEN statement
   - REVERSED phrase of the OPEN statement
   - OF/IN phrase of the COPY statement

   See Level 2 Restrictions below.

   In the following list, the shorthand notation describing module levels is shown in parentheses. For example, to summarize module information for sequential input/output, the shorthand notation is (2 SEQ 1,2).



```
   |  |  |_____ÿ Minimum level supported
   |  |________ÿ Abbreviation of module name
   |___________ÿ Supported level of elements
```

- Nucleus (2 NUC 1,2)
- Sequential I-O (1 SEQ 1,2) file.

  Level 2 restriction: the EXTEND phrase of the OPEN statement is not supported except for VSAM sequential files.

  Level 2 restriction: the REVERSED phrase of the OPEN statement does not cause file positioning, and is only applicable to tape files.

- Relative I-O (2 REL 0,2)
- Indexed I-O (2 INX 0,2)
- Sort-Merge (1 SRT 0,1)
- Inter-Program Communication (2 IPC 1,2)
- Source Text Manipulation (1 STM 0,2)

  Level 2 restriction: the OF/IN phrase of the COPY statement is treated as documentation.

In addition, the following levels of optional modules are supported:

- Intrinsic Functions (1 ITR 0,1)
- Debug (1 DEB 0,2)
- Segmentation (2 SEG 0,2)

The following optional modules of the standard are not supported:

- Report Writer
- Communications
- Debug (2 DEB 0,2)

3. FIPS Publication 21-3, Federal Information Processing Standard 21-3 , COBOL high subset.

4. International Reference Version of the ISO 7-bit code defined in *International Standard 646, 7-Bit Coded Character Set for Information Processing Interchange*.

5. The 7-bit coded character sets defined in *American National Standard X3.4-1977, Code for Information Interchange*.

Under CICS, the following language elements of X3.23-1985 are not supported:

ACCEPT
CLOSE
DELETE

DISPLAY
MERGE
OPEN
READ
RERUN
REWRITE
SORT which requires COBOL to perform I/O
START
STOP 'literal'
WRITE
USE declaratives (except USE FOR DEBUGGING)
ENVIRONMENT DIVISION and FILE SECTION when entries relate to data management.

NOTES:

The term "COBOL 85 Standard" is used in this book to refer to the combination of the following standards:

° ISO 1989:1985, Programming Languages - COBOL

ISO 1989/Amendment 1, Programming Languages - COBOL - Amendment 1: Intrinsic function module.

° X3.23-1985, American National Standard for Information Systems - Programming Language - COBOL.

X3.23a-1989, American National Standard for Information Systems - Programming Language - Intrinsic Function Module for COBOL.

The term "COBOL 74 Standard" is used in this book to refer to the following standards:

° X3.23-1974, American National Standard for Information Systems - Programming Language - COBOL.

° ISO 1989:1978, Programming Languages - COBOL

Note: The ISO Standards are equivalent to the American National Standards.

The following options are **required** to support the above standards:

Compiler Options LE/VSE Run-Time Options

ADV	AIXBLD
DYNAM	TRAP(ON)
FLAGSTD(H)	
LIB	
NOCMPR2	
NOCURRENCY	
NODBCS	
NOFASTSRT	
NOFLAGMIG	
NOFLAGSAA	
NONUMBER	
NOSEQUENCE	

```
NUMPROC(NOPFD) or
  NUMPROC(MIG)
QUOTE
TRUNC(STD)
NOWORD
ZWB
```

The following LE/VSE run-time options are used in support of the standards: UPSI, DEBUG, and NODEBUG.

APPENDIX1.10 Appendix J. Preventing File Status 39 for SAM Files

This appendix provides guidelines to help prevent common File Status 39 problems for SAM files, which are due to mismatches in the attributes for file organization, record format (fixed or variable), record length, or the code set.

For SAM ESDS files opened for INPUT or I-O, the VSAM catalog is checked to determine the record format and record length. These are checked against the values specified from the FD entry and record descriptions.

Processing Existing Files: When your program processes an existing file, code the description of the file in your COBOL program to be consistent with the file attributes of the file, for example:

Format-V Files or Format-S Files	The maximum record length specified in your program must be exactly 4 bytes smaller than the length attribute of the file.
Format-F Files	The record length specified in your program must exactly match the length attribute of the file.
Format-U Files	The maximum record length specified in your program must exactly match the length attribute of the file.

For details on how record lengths are determined from the FD entry and record descriptions in your program, see *COBOL/VSE Programming Guide*.

Defining Variable-Length Records: The easiest way to define variable-length records in your program is to use RECORD IS VARYING FROM integer-1 TO integer-2 in the FD entry and specify an appropriate value for integer-2. For example, assume that you have determined the length attribute of the file to be 104 (LRECL=104). Keeping in mind that the maximum record length is determined from the RECORD IS VARYING clause (in which values are specified) and not from the level-01 record descriptions, you could define a format-V file in your program with this code:

```
    FILE SECTION.
    FD  COMMUTER-FILE-MST
        RECORDING MODE IS V
        RECORD IS VARYING FROM 4 TO 100 CHARACTERS.
    01  COMMUTER-RECORD-A           PIC X(4).
    01  COMMUTER-RECORD-B           PIC X(75).
```

Assume that the existing file in the previous example was format-U instead of format-V. If the 104 bytes are all user data, you could define the file in your program with this code:

```
FILE SECTION.
FD  COMMUTER-FILE-MST
    RECORDING MODE IS U
    RECORD IS VARYING FROM 4 TO 104 CHARACTERS.
01  COMMUTER-RECORD-A          PIC X(4).
01  COMMUTER-RECORD-B          PIC X(75).
```

Defining Fixed-Length Records: To define fixed-length records in your program, use either the RECORD CONTAINS integer clause, or omit this clause and specify all level-01 record descriptions to be the same fixed size. In either case, use a value that equals the value of the length attribute of the file. When you intend to use the same program to process different files at execution and the files have differing fixed-length record lengths, the recommended way to avoid record-length conflicts is to code RECORD CONTAINS 0.

Converting Existing Files that Do Not Match the COBOL Record: You can re-allocate a new file with the matching LRECL, copy the data from an existing file to the new file, then use the new file as input.

Processing New Files: When your COBOL program will write records to a new file which is made available before the program is run, ensure that the file attributes you specify in the DLBL statement or the allocation do not conflict with the attributes you have specified in your program. In most cases, you only need to specify a minimum of parameters when predefining your files, as illustrated in the following example of a DLBL statement related to the FILE-CONTROL and FD entries in your program:

JCL DLBL Statement:

```
        1
        ÿ
        ÿ
// DLBL OUTFILE,'MYFILE.OUT171',0,VSAM



/*
```

COBOL/VSE Program Code:

```
ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
    FILE-CONTROL.


      SELECT CARPOOL
          ASSIGN TO OUTFILE           1
          ORGANIZATION IS SEQUENTIAL.
    .
    .
    .
```

```
   DATA DIVISION.
     FILE SECTION.

       FD CARPOOL                2
         LABEL RECORD STANDARD
         BLOCK 0 RECORDS
         RECORD 80 CHARACTERS
```

1 The filename in the DLBL statement corresponds to the assignment-name in the ASSIGN clause:

```
   // DLBL OUTFILE,'MYFILE.OUT171',0,VSAM
```

This assignment-name points to the filename of OUTFILE in the DLBL statement.

```
             ASSIGN TO OUTFILE
```

2 When you specify a file in your COBOL FILE-CONTROL entry, the file must be described in an FD entry for file-name.

```
             SELECT CARPOOL

             FD CARPOOL
```

When you do need to explicitly specify a length attribute for the file (for example, for IDCAMS define of SAM ESDS files):

- For format-V and format-S files, specify a length attribute that is 4 bytes larger than what is defined in the program.
- For format-F and format-U files, specify a length attribute that is the same as what is defined in the program.
- If you open your file as OUTPUT and write it to a printer, the compiler might add one byte to the record length to account for the carriage control character, depending on the ADV compiler option and the COBOL language used in your program. In such a case, take the added byte into account when specifying the LRECL.

For example, if your program contains the following code for a file with variable-length records:

```
    FILE SECTION.
      FD  COMMUTER-FILE-MST
          RECORDING MODE IS V
          RECORD CONTAINS 10 TO 50 CHARACTERS.
      01  COMMUTER-RECORD-A           PIC X(10).
      01  COMMUTER-RECORD-B           PIC X(50).
```

The LRECL for the file should be 54.

BACK_1 Bibliography

Subtopics:

BACK_1.1 Language Environment Publications

IBM Language Environment for VSE/ESA

Fact Sheet, GC26-8062

Concepts Guide, GC26-8063

Installation and Customization Guide, SC26-8064

Programming Guide, SC26-8065

Debugging Guide and Run-Time Messages, SC26-8066

Diagnosis Guide, SC26-8060

Licensed Program Specifications, GC26-8061

Reference Summary, SX26-3835

BACK_1.2 LE/VSE-Conforming Language Product Publications

IBM COBOL for VSE/ESA

General Information, GC26-8068

Migration Guide, GC26-8070

Installation and Customization Guide, SC26-8071

Programming Guide, SC26-8072

Language Reference, SC26-8073

Reference Summary, SX26-3834

Diagnosis Guide, SC26-8528

Licensed Program Specifications, GC26-8069

IBM PL/I for VSE/ESA

Fact Sheet, GC26-8052

Programming Guide, SC26-8053

Language Reference, SC26-8054

Licensed Program Specifications, GC26-8055

Migration Guide, SC26-8056

Installation and Customization Guide, SC26-8057

Diagnosis Guide, SC26-8058

Reference Summary, SX26-3836

Compile-Time Messages and Codes, SC26-8059

BACK_1.3 Related Publications

Subtopics:

BACK_1.3.1 VS COBOL II Publications

- *General Information*, SC26-4042
- *Licensed Program Specifications*, SC26-4044
- *Migration Guide for MVS and CMS*, SC26-3151
- *Migration Guide for VSE*, SC26-3150
- *Installation and Customization for MVS*, SC26-4048
- *Installation and Customization for CMS*, SC26-4213

- *Installation and Customization for VSE*, SC26-4696
- *Application Programming Guide for MVS and CMS*, SC26-4045
- *Application Programming Guide for VSE*, SC26-4697
- *Application Programming Language Reference*, SC26-4047
- *Application Programming Reference Summary*, SX26-3721
- *Application Programming Debugging*, SC26-4049
- *Diagnosis Guide*, LY27-9523
- *Diagnosis Reference*, LY27-9522.

BACK_1.3.2 DOS/VS COBOL Publications

- *IBM VS COBOL for DOS/VSE*, GC26-3998
- *Compiler and Library: Programmer's Guide*, SC28-6478
- *Compiler and Library: Installation Material*, SC28-6479

BACK_1.3.3 Other Product Publications

CCCA/VSE

- *COBOL and CICS Command Level Conversion Aid for VSE*, SC26-8269

COBOL Report Writer

- *COBOL Report Writer Precompiler Programmer's Manual*, SC26-4301
- *COBOL Report Writer Precompiler Installation and Operation for MVS and CMS*, SC26-4302
- *COBOL Report Writer Precompiler Installation and Operation for VSE*, SC26-4302

COBOL/SF

- *COBOL Structuring Facility User's Guide and Reference*, SC34-4080

CICS/VSE

- *Application Programming Guide*, SC33-0712
- *Application Programming Reference*, SC33-0713
- *System Definition and Operations Guide*, SC33-0706
- *Resource Definition*, SC33-0708

DFSORT for VSE/ESA

- *Application Programming Guide*, SC26-7040

DL/I DOS/VS

- *Application Programming: High-Level Programming Interface*, SH24-5009
- *Application Programming: CALL and RQDLI Interfaces*, SH12-5411

SQL/DS

- *Application Programming for VSE*, SH09-8098

BACK_1.4 Softcopy Publications

These collections contain the COBOL/VSE and LE/VSE-conforming language product publications:

VSE Collection, SK2T-0060

Application Development Collection, SK2T-1237

You can order these publications from Mechanicsburg through your local IBM

representative.

BACK_2 Glossary

The terms in this glossary are defined in accordance with their meaning in COBOL. These terms may or may not have the same meaning in other languages.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the following publications:

- *American National Standard Programming Language COBOL, ANSI X3.23-1985* (Copyright 1985 American National Standards Institute, Inc.), which was prepared by Technical Committee X3J4, which had the task of revising American National Standard COBOL, X3.23-1974.

- *American National Dictionary for Information Processing Systems* (Copyright 1982 by the Computer and Business Equipment Manufacturers Association).

American National Standard definitions are preceded by an asterisk (*).

A

*** abbreviated combined relation condition**. The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

abend. Abnormal termination of program.

*** access mode**. The manner in which records are to be operated upon within a file.

*** actual decimal point**. The physical representation, using the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

*** alphabet-name**. A user-defined word, in the SPECIAL-NAMES paragraph of the Environment Division, that assigns a name to a specific character set and/or collating sequence.

*** alphabetic character**. A letter or a space character.

*** alphanumeric character**. Any character in the computer's character set.

alphanumeric-edited character. A character within an alphanumeric character string that contains at least one B, 0 (zero), or / (slash).

*** alphanumeric function**. A function whose value contains a string of one or more characters from the computer's character

set.

*** alternate record key**. A key, other than the prime record key, whose contents identify a record within an indexed file.

ANSI (American National Standards Institute). An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

*** argument.**. An identifier, a literal, an arithmetic expression, or a function identifier that specifies a value to be used in the evaluation of a function.

*** arithmetic expression**. An identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

*** arithmetic operation**. The process caused by the execution of an arithmetic statement, or the evaluation of an arithmetic expression, that results in a mathematically correct solution to the arguments presented.

*** arithmetic operator**. A single character, or a fixed two-character combination that belongs to the following set:

Character Meaning + addition - subtraction * multiplication / division ** exponentiation

*** arithmetic statement**. A statement that causes an arithmetic operation to be executed. The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements.

array. In LE/VSE, an aggregate consisting of data objects, each of which may be uniquely referenced by subscripting. Roughly analogous to a COBOL table.

*** ascending key**. A key upon the values of which data is ordered, starting with the lowest value of the key up to the highest value of the key, in accordance with the rules for comparing data items.

ASCII. American National Standard Code for Information Interchange. The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange between data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

Note: IBM has defined an extension to ASCII code (characters 128-255).

assignment-name. A name that identifies the organization of a COBOL file and the name by which it is known to the system.

*** assumed decimal point**. A decimal point position that does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning with no physical representation.

*** AT END condition**. A condition caused:

1. During the execution of a READ statement for a sequentially accessed file, when no next logical record exists in the file, or when the number of significant digits in the relative record number is larger than the size of the relative key data item, or when an optional input file is not present.

2. During the execution of a RETURN statement, when no next logical record exists for the associated sort or merge file.

3. During the execution of a SEARCH statement, when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

B

binary item. A numeric data item represented in binary notation (on the base 2 numbering system). Binary items have a decimal equivalent consisting of the decimal digits 0 through 9, plus an operational sign. The leftmost bit of the item is the operational sign.

binary search. A dichotomizing search in which, at each step of the search, the set of data elements is divided by two; some appropriate action is taken in the case of an odd number.

*** block**. A physical unit of data that normally contains one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either contained within the block or that overlap the block. The term is synonymous with physical record.

breakpoint. A place in a computer program, usually specified by an instruction, where its execution may be interrupted by external intervention or by a monitor program.

buffer. A portion of storage used to hold input or output data temporarily.

byte. A string consisting of a certain number of bits, usually eight, treated as a unit, and representing a character.

C

callable services. In LE/VSE, a set of services that can be invoked by a COBOL program using the conventional LE/VSE-defined call interface, and usable by all programs sharing the LE/VSE conventions.

called program. A program that is the object of a CALL statement.

*** calling program**. A program that executes a CALL to another program.

case structure. A program processing logic in which a series of conditions is tested in order to make a choice between a number of resulting actions.

*** character**. The basic indivisible unit of the language.

character position. The amount of physical storage required to store a single standard data format character described as USAGE IS DISPLAY.

character set. All the valid characters for a programming language or a computer system.

*** character string**. A sequence of contiguous characters that form a COBOL word, a literal, a PICTURE character string, or a comment-entry. Must be delimited by separators.

checkpoint. A point at which information about the status of a job and the system can be recorded so that the job step can be later restarted.

*** class condition**. The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic, is wholly numeric, or consists exclusively of those characters listed in the definition of a class-name.

*** class-name**. A user-defined word defined in the SPECIAL-NAMES paragraph of the Environment Division that assigns a

name to the proposition for which a truth value can be defined, that the content of a data item consists exclusively of those characters listed in the definition of the class-name.

*** clause**. An ordered set of consecutive COBOL character strings whose purpose is to specify an attribute of an entry.

CMS (Conversational Monitor System). A virtual machine operating system that provides general interactive, time-sharing, problem solving, and program development capabilities, and that operates only under the control of the VM/SP control program.

*** COBOL character set**. The complete COBOL character set consists of the characters listed below:

Character Meaning 0,1...,9 digit A,B,...,Z uppercase letter a,b,...,z lowercase letter º space + plus sign - minus sign (hyphen) * asterisk / slant (virgule, slash) = equal sign $ currency sign , comma (decimal point) ; semicolon . period (decimal point, full stop) " quotation mark (left parenthesis) right parenthesis > greater than symbol < less than symbol : colon

*** COBOL word**. See "word."

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

*** column**. A character position within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

*** combined condition**. A condition that is the result of connecting two or more conditions with the AND or the OR logical operator.

*** comment-entry**. An entry in the Identification Division that may be any combination of characters from the computer's character set.

*** comment line**. A source program line represented by an asterisk (*) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a slant (/) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line causes page ejection prior to printing the comment.

*** common program**. A program which, despite being directly contained within another program, may be called from any program directly or indirectly contained in that other program.

*** compile**. (1) To translate a program expressed in a high-level language into a program expressed in an intermediate language, assembly language, or a computer language. (2) To prepare a machine language program from a computer program written in another programming language by making use of the overall logic structure of the program, or generating more than one computer instruction for each symbolic statement, or both, as well as performing the function of an assembler.

*** compile time**. The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL object program.

compiler. A program that translates a program written in a higher level language into a machine language object program.

compiler directing statement. A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation. The SAA(*) compiler directing statements are COPY, EJECT, SKIP1/2/3, TITLE, and USE. The non-SAA compiler-directing statements are: REPLACE, BASIS, INSERT, and DELETE.

*** complex condition**. A condition in which one or more logical operators act upon one or more conditions. (See also "negated simple condition," and "combined condition," "negated combined condition.")

*** computer-name**. A system-name that identifies the computer upon which the program is to be compiled or run.

*** condition**. A status of a program at run time for which a truth value can be determined. Where the term 'condition' (condition-1, condition-2,...) appears in these language specifications in or in reference to 'condition' (condition-1, condition-2,...) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesized, or a

combined condition consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

*** conditional expression**. A simple condition or a complex condition specified in an EVALUATE, IF, PERFORM, or SEARCH statement. (See also "simple condition" and "complex condition.")

*** conditional phrase**. A conditional phrase specifies the action to be taken upon determination of the truth value of a condition resulting from the execution of a conditional statement.

*** conditional statement**. A statement specifying that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

*** conditional variable**. A data item one or more values of which has a condition-name assigned to it.

*** condition-name**. A user-defined word that assigns a name to a subset of values that a conditional variable may assume; or a user-defined word assigned to a status of an implementor defined switch or device. When 'condition-name' is used in the general formats, it represents a unique data item reference consisting of a syntactically correct combination of a 'condition-name', together with qualifiers and subscripts, as required for uniqueness of reference.

*** condition-name condition**. The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

*** Configuration Section**. A section of the Environment Division that describes overall specifications of source and object programs.

CONSOLE. A COBOL environment-name associated with the operator console.

*** contiguous items**. Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to each other.

*** counter**. A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

cross-reference listing. The portion of the compiler listing that contains information on where files, fields, and indicators are defined, referenced, and modified in a program.

currency sign. The character '$' of the COBOL character set or that character defined by the CURRENCY compiler option. IF the NOCURRENCY compiler option is in effect, the currency sign is defined as the character '$'.

currency symbol. The character defined by the CURRENCY compiler option or by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If the NOCURRENCY compiler option is in effect for a COBOL source program and the CURRENCY SIGN clause is also **not** present in the the source program, the currency symbol is identical to the currency sign.

*** current record**. In file processing the record that is available in the record area associated with a file.

*** current volume pointer**. A conceptual entity that points to the current volume of a sequential file.

D

*** data clause**. A clause, appearing in a data description entry in the Data Division of a COBOL program, that provides information describing a particular attribute of a data item.

*** data description entry**. An entry, in the Data Division of a COBOL program, that contains a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

Data Division. One of the four main components of a COBOL program. The Data Division describes the data to be processed by the object program: files to be used and the records contained within them; internal Working-Storage records that will be needed; data to be made available in more than one program in the COBOL run unit.

*** data item**. A unit of data (excluding literals) defined by a COBOL program or by the rules for function evaluation.

*** data-name**. A user-defined word that names a data item described in a data description entry. When used in the general formats, 'data-name' represents a word that must not be reference-modified, subscripted or qualified unless specifically permitted by the rules for the format.

DBCS (Double-Byte Character Set). See "Double-Byte Character Set (DBCS)."

*** debugging line**. A debugging line is any line with a 'D' in the indicator area of the line.

*** debugging section**. A section that contains a USE FOR DEBUGGING statement.

*** declarative sentence**. A compiler directing sentence consisting of a single USE statement terminated by the separator period.

*** declaratives**. A set of one or more special purpose sections, written at the beginning of the Procedure Division, the first of which is preceded by the keyword DECLARATIVES and the last of which is followed by the keywords END DECLARATIVES. A declarative contains a section header, followed by a USE compiler directing sentence, followed by a set of zero, one, or more associated paragraphs.

*** de-edit**. The logical removal of all editing characters from a numeric edited data item in order to determine that item's unedited numeric value.

*** delimited scope statement**. Any statement that includes its explicit scope terminator.

*** delimiter**. A character or a sequence of contiguous characters that identify the end of a string of characters and separate that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

*** descending key**. A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

digit. Any of the numerals from 0 through 9. In COBOL, the term is not used in reference to any other symbol.

*** digit position**. The amount of physical storage required to store a single digit. This amount may vary depending on the usage specified in the data description entry that defines the data item.

*** direct access**. The facility to obtain data from storage devices or to enter data into a storage device in such a way that the process depends only on the location of that data and not on a reference to data previously accessed.

*** division**. A collection of zero, one or more sections or paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. Each division consists of the division header and the related division body. There are four (4) divisions in a COBOL program: Identification, Environment, Data, and Procedure.

*** division header**. A combination of words followed by a separator period that indicates the beginning of a division. The division headers in a COBOL program are:

IDENTIFICATION DIVISION. ENVIRONMENT DIVISION. DATA DIVISION. PROCEDURE DIVISION.

Double-Byte Character Set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require Double-Byte Character Sets. Since each character requires two bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software which are DBCS-capable.

*** dynamic access**. An access mode in which specific logical records can be obtained from or placed into a mass storage file in a nonsequential manner and obtained from a file in a sequential manner during the scope of the same OPEN statement.

Dynamic Storage Area (DSA). Dynamically acquired storage containing a register save area and an area available for dynamic storage allocation (such as program variables). DSAs are generally allocated within STACK segments managed by LE/VSE.

E

*** EBCDIC (Extended Binary-Coded Decimal Interchange Code)**. A coded character set consisting of 8-bit coded characters.

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC (Extended Binary-Coded-Decimal Interchange Code) set.

edited data item. A data item that has been modified by suppressing zeros and/or inserting editing characters.

*** editing character**. A single character or a fixed two-character combination belonging to the following set:

Character Meaning º space 0 zero + plus - minus CR credit DB debit Z zero suppress * check protect $ currency sign , comma (decimal point) . period (decimal point) / slant (virgule, slash)

element (text element). One logical unit of a string of text, such as the description of a single data item or verb, preceded by a unique code identifying the element type.

*** elementary item**. A data item that is described as not being further logically subdivided.

enclave. In LE/VSE, an independent collection of routines, one of which is designated as the *main* program. An enclave is roughly analogous to a COBOL program which contains called programs.

*** end of Procedure Division**. The physical position of a COBOL source program after which no further procedures appear.

*** end program header**. A combination of words, followed by a separator period, that indicates the end of a COBOL source program. The end program header is:

```
END PROGRAM program-name.
```

*** entry**. Any descriptive set of consecutive clauses terminated by a separator period and written in the Identification Division, Environment Division, or Data Division of a COBOL program.

*** environment clause**. A clause that appears as part of an Environment Division entry.

Environment Division. One of the four main component parts of a COBOL program. The Environment Division describes the computers upon which the source program is compiled and those on which the object program is executed, and provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored.

environment-name. A name, specified by IBM, that identifies system logical units, printer and card punch control characters, report codes, and/or program switches. Valid environment-names for SAA COBOL are SYSIN, SYSOUT, CONSOLE, C01, CSP, and UPSI-0 through UPSI-7. When an environment-name is associated with a mnemonic-name in the Environment Division, the mnemonic-name may then be substituted in any format in which such substitution is valid.

execution time. See "run time."

execution-time environment. See "run-time environment."

*** explicit scope terminator**. A reserved word which terminates the scope of a particular Procedure Division statement.

exponent. A number, indicating the power to which another number (the base) is to be raised. Positive exponents denote multiplication, negative exponents denote division, fractional exponents denote a root of a quantity. In COBOL, an exponential expression is indicated with the symbol '**' followed by the exponent.

*** expression**. An arithmetic or conditional expression.

*** extend mode**. The state of a file after execution of an OPEN statement, with the EXTEND phrase specified for that file, and before the execution of a CLOSE statement, without the REEL or UNIT phrase for that file.

*** external data**. The data described in a program as external data items and external file connectors.

*** external data item**. A data item which is described as part of an external record in one or more programs of a run unit and which itself may be referenced from any program in which it is described.

*** external data record**. A logical record which is described in one or more programs of a run unit and whose constituent data items may be referenced from any program in which they are described.

external decimal item. A format for representing numbers in which the digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the rightmost byte. Bits 0 through 3 of all other bytes contain 1's (hex F). For example, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. (Also know as "zoned decimal item.")

*** external file connector**. A file connector which is accessible to one or more object programs in the run unit.

*** external switch**. A hardware or software device, defined and named by the implementor, which is used to indicate that one of two alternate states exists.

F

*** figurative constant**. A compiler-generated value referenced through the use of certain reserved words.

*** file**. A collection of logical records.

*** file attribute conflict condition**. An unsuccessful attempt has been made to execute an input-output operation on a file and the file attributes, as specified for that file in the program, do not match the fixed attributes for that file.

*** file clause**. A clause that appears as part of any of the following Data Division entries: file description entry (FD entry) and sort-merge file description entry (SD entry).

*** file connector**. A storage area which contains information about a file and is used as the linkage between a file-name and a physical file and between a file-name and its associated record area.

File-Control. The name of an Environment Division paragraph in which the data files for a given source program are declared.

*** file control entry**. A SELECT clause and all its subordinate clauses which declare the relevant physical attributes of a file.

*** file description entry**. An entry in the File Section of the Data Division that contains the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

*** file-name**. A user-defined word that names a file connector described in a file description entry or a sort-merge file

description entry within the File Section of the Data Division.

*** file organization**. The permanent logical file structure established at the time that a file is created.

***file position indicator**. A conceptual entity that contains the value of the current key within the key of reference for an indexed file, or the record number of the current record for a sequential file, or the relative record number of the current record for a relative file, or indicates that no next logical record exists, or that an optional input file is not present, or that the at end condition already exists, or that no valid next record has been established.

*** file section**. The section of the Data Division that contains file description entries and sort-merge file description entries together with their associated record descriptions.

*** fixed file attributes**. Information about a file which is established when a file is created and cannot subsequently be changed during the existence of the file. These attributes include the organization of the file (sequential, relative, or indexed), the prime record key, the alternate record keys, the code set, the minimum and maximum record size, the record type (fixed or variable), the collating sequence of the keys for indexed files, the blocking factor, the padding character, and the record delimiter.

*** fixed length record**. A record associated with a file whose file description or sort-merge description entry requires that all records contain the same number of character positions.

fixed-point number. A numeric data item defined with a PICTURE clause that specifies the location of an optional sign, the number of digits it contains, and the location of an optional decimal point. The format may be either binary, packed decimal, or external decimal.

floating-point number. A numeric data item containing a fraction and an exponent. Its value is obtained by multiplying the fraction by the base of the numeric data item raised to the power specified by the exponent.

*** format**. A specific arrangement of a set of data.

*** function**. A temporary data item whose value is determined at the time the function is referenced during the execution of a statement.

*** function identifier**. A syntactically correct combination of character strings and separators that references a function. The data item represented by a function is uniquely identified by a function-name with its arguments, if any. A function identifier may include a reference modifier. A function identifier that references an alphanumeric function may be specified anywhere in the general formats that an identifier may be specified, subject to certain restrictions. A function identifier that references an integer or numeric function may be referenced anywhere in the general formats that an arithmetic expression may be specified.

function-name. A word that names the mechanism whose invocation, along with required arguments, determines the value of a function.

G

*** global name**. A name which is declared in only one program but which may be referenced from that program and from any program contained within that program. Condition-names, data-names, file-names, record-names, report-names, and some special registers may be global names.

*** group item**. A data item that contains subordinate data items.

H

header label. (1) A file label that precedes the data records on a unit of recording media. (2) Synonym for beginning-of-file label.

*** high order end**. The leftmost character of a string of characters.

I

IBM COBOL extension. Certain COBOL syntax and semantics supported by IBM compilers in addition to those described in ANSI Standard.

Identification Division. One of the four main component parts of a COBOL program. The Identification Division identifies the source program and the object program. The Identification Division may include the following documentation: author name, installation, or date.

*** identifier**. A syntactically correct combination of character strings and separators that names a data item. When referencing a data item which is not a function, an identifier consists of a data-name, together with its qualifiers, subscripts, and reference modifier, as required for uniqueness of reference. When referencing a data item which is a function, a function identifier is used.

*** imperative statement**. A statement that either begins with an imperative verb and specifies an unconditional action to be taken or is a conditional statement that is delimited by its explicit scope terminator (delimited scope statement). An imperative statement may consist of a sequence of imperative statements.

*** implicit scope terminator**. A separator period which terminates the scope of any preceding unterminated statement, or a phrase of a statement which by its occurrence indicates the end of the scope of any statement contained within the preceding phrase.

*** index**. A computer storage area or register, the content of which represents the identification of a particular element in a table.

*** index data item**. A data item in which the values associated with an index-name can be stored in a form specified by the implementor.

indexed data-name. An identifier that contains a data-name, followed by one or more index-names enclosed in parentheses.

*** indexed file**. A file with indexed organization.

*** indexed organization**. The permanent logical file structure in which each record is identified by the value of one or more keys within that record.

indexing. Synonymous with subscripting using index-names.

*** index-name**. A user-defined word that names an index associated with a specific table.

*** initial program**. A program that is placed into an initial state every time the program is called in a run unit.

*** initial state**. The state of a program when it is first called in a run unit.

*** input file**. A file that is opened in the INPUT mode.

*** input mode**. The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement, without the REEL or UNIT phrase for that file.

*** input-output file**. A file that is opened in the I-O mode.

*** Input-Output Section**. The section of the Environment Division that names the files and the external media required by an object program and that provides information required for transmission and handling of data during execution of the object program.

*** Input-Output statement**. A statement that causes files to be processed by performing operations upon individual records or upon the file as a unit. The input-output statements are: ACCEPT (with the identifier phrase), CLOSE, DELETE, DISABLE, DISPLAY, ENABLE, OPEN, PURGE, READ, RECEIVE, REWRITE, SEND, SET (with the TO ON or TO OFF phrase), START, and WRITE.

*** input procedure**. A set of statements, to which control is given during the execution of a SORT statement, for the purpose of controlling the release of specified records to be sorted.

*** integer**. (1) A numeric literal that does not include any digit positions to the right of the decimal point.

(2) A numeric data item defined in the Data Division that does not include any digit positions to the right of the decimal point.

(3) A numeric function whose definition provides that all digits to the right of the decimal point are zero in the returned value for any possible evaluation of the function.

integer function. A function whose category is numeric and whose definition does not include any digit positions to the right of the decimal point.

intermediate result. An intermediate field containing the results of a succession of arithmetic operations.

*** internal data**. The data described in a program excluding all external data items and external file connectors. Items described in the Linkage Section of a program are treated as internal data.

*** internal data item**. A data item which is described in one program in a run unit. An internal data item may have a global name.

internal decimal item. A format in which each byte in a field except the rightmost byte represents two numeric digits. The rightmost byte contains one digit and the sign. For example, the decimal value +123 is represented as 0001 0010 0011 1111. (Also known as packed decimal.)

*** internal file connector**. A file connector which is accessible to only one object program in the run unit.

*** intra-record data structure**. The entire collection of groups and elementary data items from a logical record which is defined by a contiguous subset of the data description entries which describe that record. These data description entries include all entries whose level-number is greater than the level-number of the first data description entry describing the intra-record data structure.

*** invalid key condition**. A condition, at object time, caused when a specific value of the key associated with an indexed or relative file is determined to be invalid.

*** I-O-CONTROL**. The name of an Environment Division paragraph in which object program requirements for rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device are specified.

*** I-O-CONTROL entry**. An entry in the I-O-CONTROL paragraph of the Environment Division which contains clauses which provide information required for the transmission and handling of data on named files during the execution of a program.

*** I-O-Mode**. The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before

the execution of a CLOSE statement without the REEL or UNIT phase for that file.

*** I-O status**. A conceptual entity which contains the two-character value indicating the resulting status of an input-output operation. This value is made available to the program through the use of the FILE STATUS clause in the file control entry for the file.

iteration structure. A program processing logic in which a series of statements is repeated while a condition is true or until a condition is true.

K

K. When referring to storage capacity, two to the tenth power; 1024 in decimal notation.

*** key**. A data item that identifies the location of a record, or a set of data items which serve to identify the ordering of data.

*** key of reference**. The key, either prime or alternate, currently being used to access records within an indexed file.

*** keyword**. A reserved word or function-name whose presence is required when the format in which the word appears is used in a source program.

kilobyte (KB). One kilobyte equals 1024 bytes.

L

*** language-name**. A system-name that specifies a particular programming language.

LE-conforming. A characteristic of compiler products which produce object code conforming to the Language Environment format.

*** letter**. A character belonging to one of the following two sets:

1. Uppercase letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

2. Lowercase letters: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z.

*** level indicator**. Two alphabetic characters that identify a specific type of file or a position in a hierarchy. The level indicators in the Data Division are: CD, FD, and SD.

*** level-number**. A user-defined word, expressed as a two digit number, which indicates the hierarchical position of a data item or the special properties of a data description entry. Level-numbers in the range from 1 through 49 indicate the position of a data item in the hierarchic structure of a logical record. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-numbers 66, 77 and 88 identify special properties of a data description entry.

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

*** library text**. A sequence of text words, comment lines, the separator space, or the separator pseudo-text delimiter in a COBOL library.

*** LINAGE-COUNTER**. A special register whose value points to the current position within the page body.

Linkage Section. The section in the Data Division of the called program that describes data items available from the calling program. These data items may be referred to by both the calling and called program.

literal. A character string whose value is specified either by the ordered set of characters comprising the string, or by the use of a figurative constant.

*** logical operator**. One of the reserved words AND, OR, or NOT. In the formation of a condition, either AND, or OR, or both can be used as logical connectives. NOT can be used for logical negation.

*** logical record**. The most inclusive data item. The level-number for a record is 01. A record may be either an elementary item or a group of items. The term is synonymous with record.

*** low order end**. The rightmost character of a string of characters.

M

main program. In a hierarchy of programs and subroutines, the first program to receive control when the programs are run.

*** mass storage**. A storage medium in which data may be organized and maintained in both a sequential and nonsequential manner.

*** mass storage device**. A device having a large storage capacity; for example, magnetic disk, magnetic drum.

*** mass storage file**. A collection of records that is assigned to a mass storage medium.

*** megabyte (M)**. One megabyte equals 1,048,576 bytes.

*** merge file**. A collection of records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

*** mnemonic-name**. A user-defined word that is associated in the Environment Division with a specified implementor-name.

multitasking. Mode of operation that provides for the concurrent, or interleaved, execution of two or more tasks. In LE/370, synonymous with *multithreading*.

N

name. A word containing not more than 30 characters that defines a COBOL operand.

*** native character set**. The implementor-defined character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

*** native collating sequence**. The implementor-defined collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

*** negated combined condition**. The 'NOT' logical operator immediately followed by a parenthesized combined condition.

*** negated simple condition**. The 'NOT' logical operator immediately followed by a simple condition.

nested program. A program that is directly contained within another program.

*** next executable sentence**. The next sentence to which control will be transferred after execution of the current statement is complete.

*** next executable statement**. The next statement to which control will be transferred after execution of the current statement is complete.

*** next record**. The record that logically follows the current record of a file.

*** noncontiguous items**. Elementary data items in the Working-Storage and Linkage Sections that bear no hierarchic relationship to other data items.

*** nonnumeric item**. A data item whose description permits its content to contain any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

*** nonnumeric literal**. A literal bounded by quotation marks. The string of characters may include any character in the computer's character set.

null. Figurative constant used to assign the value of an invalid address to pointer data items. NULLS can be used wherever NULL can be used.

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

numeric-edited item. A numeric item that is in such a form that it may be used in printed output. It may consist of external decimal digits from 0 through 9, the decimal point, commas, the dollar sign, editing sign control symbols, plus other editing symbols.

*** numeric function**. A function whose class and category are numeric but which for some possible evaluation does not satisfy the requirements of integer functions.

*** numeric item**. A data item whose description restricts its content to a value represented by characters chosen from the digits from '0' through '9'; if signed, the item may also contain a '+', '-', or other representation of an operational sign.

*** numeric literal**. A literal containing one or more numeric characters that also contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

O

object code. Output from a compiler or assembler which is itself executable machine code or is suitable for processing to produce executable machine code.

*** OBJECT-COMPUTER**. The name of an Environment Division paragraph in which the computer environment, within which the object program is executed, is described.

*** object computer entry**. An entry in the OBJECT-COMPUTER paragraph of the Environment Division which contains clauses which describe the computer environment in which the object program is to be executed.

object deck. A portion of an object program suitable as input to a linkage editor. Synonymous with *object module* and *text deck*.

object module. Synonym for *object deck* or *text deck*.

*** object of entry**. A set of operands and reserved words, within a Data Division entry of a COBOL program, that immediately follows the subject of the entry.

*** object program**. A set or group of executable machine language instructions and other material designed to interact with data to provide problem solutions. In this context, an object program is generally the machine language result of the operation of a COBOL compiler on a source program. Where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'object program.'

*** object time**. The time at which an object program is executed. The term is synonymous with execution time.

*** obsolete element**. A COBOL language element in Standard COBOL that is to be deleted from the next revision of Standard COBOL.

*** open mode**. The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O or EXTEND.

*** operand**. Whereas the general definition of operand is "that component which is operated upon," for the purposes of this document, any lowercase word (or words) that appears in a statement or entry format may be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

*** operational sign**. An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

*** optional file**. A file which is declared as being not necessarily present each time the object program is executed. The object program causes an interrogation for the presence or absence of the file.

*** optional word**. A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

OS/2 (Operating System/2). A multi-tasking operating system for the IBM Personal Computer family that allows you to run both DOS mode and OS/2 mode programs.

*** output file**. A file that is opened in either the OUTPUT mode or EXTEND mode.

*** output mode**. The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified, for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file.

*** output procedure**. A set of statements to which control is given during execution of a SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function reaches a point at which it can select the next record in merged order when requested.

overflow condition. A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

P

packed decimal item. See "internal decimal item."

*** padding character**. An alphanumeric character used to fill the unused character positions in a physical record.

page. A vertical division of output data representing a physical separation of such data, the separation being based on internal logical requirements and/or external characteristics of the output medium.

*** page body**. That part of the logical page in which lines can be written and/or spaced.

*** paragraph**. In the Procedure Division, a paragraph-name followed by a separator period and by zero, one, or more sentences. In the Identification and Environment Divisions, a paragraph header followed by zero, one, or more entries.

*** paragraph header**. A reserved word, followed by the separator period, that indicates the beginning of a paragraph in the Identification and Environment Divisions. The permissible paragraph headers in the Identification Division are:

```
PROGRAM-ID. AUTHOR. INSTALLATION. DATE-WRITTEN. DATE-COMPILED. SECURITY.
```

The permissible paragraph headers in the Environment Division are:

```
SOURCE-COMPUTER. OBJECT-COMPUTER. SPECIAL-NAMES. FILE-CONTROL. I-O-CONTROL.
```

*** paragraph-name**. A user-defined word that identifies and begins a paragraph in the Procedure Division.

parameter. Parameters are used to pass data values between calling and called programs.

password. A unique string of characters that a program, computer operator, or user must supply to meet security requirements before gaining access to data.

*** phrase**. A phrase is an ordered set of one or more consecutive COBOL character strings that form a portion of a COBOL procedural statement or of a COBOL clause.

*** physical record**. See "block."

pointer data item. A data item in which address values can be stored. Data items are explicitly defined as pointers with the USAGE IS POINTER clause. ADDRESS OF special registers are implicitly defined as pointer data items. Pointer data items can be compared for equality or moved to other pointer data items.

portability. The ability to transfer an application program from one application platform to another with relatively few changes to the source program.

*** prime record key**. A key whose contents uniquely identify a record within an indexed file.

*** priority-number**. A user-defined word which classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers may contain only the characters '0','1', ... , '9'. A segment-number may be expressed either as a one or two digit number.

*** procedure**. A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

*** procedure branching statement**. A statement that causes the explicit transfer of control to a statement other than the next executable statement in the sequence in which the statements are written in the source program. The procedure branching statements are: ALTER, CALL, EXIT, EXIT PROGRAM, GO TO, MERGE, (with the OUTPUT PROCEDURE phrase), PERFORM and SORT (with the INPUT PROCEDURE or OUTPUT PROCEDURE phrase).

Procedure Division. One of the four main component parts of a COBOL program. The Procedure Division contains instructions for solving a problem. The Procedure Division may contain imperative statements, conditional statements, compiler directing statements, paragraphs, procedures, and sections.

*** procedure-name**. A user-defined word that is used to name a paragraph or section in the Procedure Division. It consists of a paragraph-name (which may be qualified), or a section-name.

procedure-pointer data item. A data item in which a pointer to an entry point can be stored. A data item defined with the USAGE IS PROCEDURE-POINTER clause contains the address of a procedure entry point.

*** program identification entry**. An entry in the PROGRAM-ID paragraph of the Identification Division which contains clauses that specify the program-name and assign selected program attributes to the program.

*** program-name**. In the Identification Division and the end program header, a user-defined word that identifies a COBOL source program.

*** pseudo-text**. A sequence of text words, comment lines, or the separator space in a source program or COBOL library bounded by, but not including, pseudo-text delimiters.

*** pseudo-text delimiter**. Two contiguous equal sign characters (==) used to delimit pseudo-text.

*** punctuation character**. A character that belongs to the following set:

Character Meaning

, comma ; semicolon : colon . period (full stop) " quotation mark (left parenthesis) right parenthesis º space = equal sign

---

Q

*** qualified data-name**. An identifier that contains a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

*** qualifier**.

1. A data-name or a name associated with a level indicator which is used in a reference either together with another data-name which is the name of an item that is subordinate to the qualifier or together with a condition-name.

2. A section-name that is used in a reference together with a paragraph-name specified in that section.

3. A library-name that is used in a reference together with a text-name associated with that library.

R

*** random access**. An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from, or placed into a relative or indexed file.

*** record**. See "logical record."

*** record area**. A storage area allocated for the purpose of processing the record described in a record description entry in the File Section of the Data Division. In the File Section, the current number of character positions in the record area is determined by the explicit or implicit RECORD clause.

*** record description**. See "record description entry."

*** record description entry**. The total set of data description entries associated with a particular record. The term is synonymous with record description.

recording mode. The format of the logical records in a file. Recording mode can be F (fixed-length), V (variable-length), S (spanned), or U (undefined).

record key. A key whose contents identify a record within an indexed file. Within an indexed file in SAA COBOL, a record key is the prime record key.

*** record-name**. A user-defined word that names a record described in a record description entry in the Data Division of a COBOL program.

*** record number**. The ordinal number of a record in the file whose organization is sequential.

reel. A discrete portion of a storage medium, the dimensions of which are determined by each implementor, that contains part of a file, all of a file, or any number of files. The term is synonymous with unit and volume.

reentrant. The attribute of a program or routine that allows more than one user to share a single copy of a phase.

*** reference format**. A format that provides a standard method for describing COBOL source programs.

reference modification. A method of defining a new alphanumeric data item by specifying the leftmost character and length relative to the leftmost character of another alphanumeric data item.

*** reference modifier**. A syntactically correct combination of character strings and separators that defines a unique data item. It includes a delimiting left parenthesis separator, the leftmost character position, a colon separator, optionally a length, and a delimiting right parenthesis separator.

*** relation**. See "relational operator." or "relation condition"

*** relational operator**. A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meanings are:

Operator Meaning IS GREATER THAN Greater than IS > Greater than IS NOT GREATER THAN Not greater than IS NOT > Not greater than

IS LESS THAN Less than IS < Less than IS NOT LESS THAN Not less than IS NOT < Not less than

IS EQUAL TO Equal to IS = Equal to IS NOT EQUAL TO Not equal to IS NOT = Not equal to

IS GREATER THAN OR EQUAL TO Greater than or equal to IS >= Greater than or equal to

IS LESS THAN OR EQUAL TO Less than or equal to IS <= Less than or equal to

*** relation character**. A character that belongs to the following set:

Character Meaning

> greater than < less than = equal to

*** relation condition**. The proposition, for which a truth value can be determined, that the value of an arithmetic expression, data item, nonnumeric literal, or index-name has a specific relationship to the value of another arithmetic expression, data item, nonnumeric literal, or index name. (See also "relational operator.")

*** relative file**. A file with relative organization.

*** relative key**. A key whose contents identify a logical record in a relative file.

*** relative organization**. The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

*** relative record number**. The ordinal number of a record in a file whose organization is relative. This number is treated as a numeric literal which is an integer.

*** reserved word**. A COBOL word specified in the list of words that may be used in a COBOL source program, but that must not appear in the program as user-defined words or system-names.

*** resource**. A facility or service, controlled by the operating system, that can be used by an executing program.

*** resultant identifier**. A user-defined data item that is to contain the result of an arithmetic operation.

routine. A set of statements in a COBOL program that causes the computer to perform an operation or series of related operations. In LE/VSE, refers to either a procedure, function, or subroutine.

*** routine-name**. A user-defined word that identifies a procedure written in a language other than COBOL.

*** run time**. The time at which an object program is executed. The term is synonymous with object time.

run-time environment. The environment in which a COBOL program executes.

*** run unit**. One or more object programs which interact with one another and which function, at object time, as an entity to provide problem solutions.

S

SAM (Sequential Access Method). An extended version of the basic sequential access method (BSAM). When this method is used, a queue is formed of input data blocks that are awaiting processing or of output data blocks that have been processed and are awaiting transfer to auxiliary storage or to an output device.

SBCS (Single Byte Character Set). See "Single Byte Character Set (SBCS)".

scope terminator. A COBOL reserved word that marks the end of certain Procedure Division statements. It may be either explicit (END-ADD, for example) or implicit (separator period).

*** section**. A set of zero, one or more paragraphs or entities, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

*** section header**. A combination of words followed by a separator period that indicates the beginning of a section in the Environment, Data, and Procedure Divisions. In the Environment and Data Divisions, a section header contains reserved words followed by a separator period. The permissible section headers in the Environment Division are:

```
CONFIGURATION SECTION. INPUT-OUTPUT SECTION.
```

The permissible section headers in the Data Division are:

```
FILE SECTION. WORKING-STORAGE SECTION. LINKAGE SECTION.
```

In the Procedure Division, a section header contains a section-name, followed by the reserved word SECTION, followed by a separator period.

*** section-name**. A user-defined word that names a section in the Procedure Division.

selection structure. A program processing logic in which one or another series of statements is executed, depending on whether a condition is true or false.

*** sentence**. A sequence of one or more statements, the last of which is terminated by a separator period.

*** separately-compiled program**. A program which, together with its contained programs, is compiled separately from all other programs.

*** separator**. A character or two contiguous characters used to delimit character strings.

*** separator comma**. A comma (,) followed by a space used to delimit character strings.

*** separator period**. A period (.) followed by a space used to delimit character strings.

*** separator semicolon**. A semicolon (;) followed by a space used to delimit character strings.

sequence structure. A program processing logic in which a series of statements is executed in sequential order.

*** sequential access**. An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of records in the file.

*** sequential file**. A file with sequential organization.

*** sequential organization**. The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

serial search. A search in which the members of a set are consecutively examined, beginning with the first member and ending with the last.

*** 77-level-description-entry**. A data description entry that describes a noncontiguous data item with the level-number 77.

*** sign condition**. The proposition, for which a truth value can be determined, that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

*** simple condition**. Any single condition chosen from the set:

Relation condition Class condition Condition-name condition Switch-status condition Sign condition

Single Byte Character Set (SBCS). A set of characters in which each character is represented by a single byte. See also "EBCDIC (Extended Binary-Coded Decimal Interchange Code)."

slack bytes. Bytes inserted between data items or records to ensure correct alignment of some numeric items. Slack bytes contain no meaningful data. In some cases, they are inserted by the compiler; in others, it is the responsibility of the programmer to insert them. The SYNCHRONIZED clause instructs the compiler to insert slack bytes when they are needed for proper alignment. Slack bytes between records are inserted by the programmer.

*** sort file**. A collection of records to be sorted by a SORT statement. The sort file is created and can be used by the sort function only.

*** sort-merge file description entry**. An entry in the File Section of the Data Division that contains the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

*** SOURCE-COMPUTER**. The name of an Environment Division paragraph in which the computer environment, within which the source program is compiled, is described.

*** source computer entry**. An entry in the SOURCE-COMPUTER paragraph of the Environment Division which contains clauses which describe the computer environment in which the source program is to be compiled.

*** source item**. An identifier designated by a SOURCE clause that provides the value of a printable item.

source program. Although it is recognized that a source program may be represented by other forms and symbols, in this document it always refers to a syntactically correct set of COBOL statements. A COBOL source program commences with the Identification Division or a COPY statement. A COBOL source program is terminated by the end program header, if specified, or by the absence of additional source program lines.

*** special character**. A character that belongs to the following set:

Character Meaning

+ plus sign - minus sign (hyphen) * asterisk / slant (virgule, slash) = equal sign $ currency sign , comma (decimal point) ; semicolon . period (decimal point, full stop) " quotation mark (left parenthesis) right parenthesis > greater than symbol < less than symbol : colon

*** special-character word**. A reserved word that is an arithmetic operator or a relation character.

SPECIAL-NAMES. The name of an Environment Division paragraph in which environment-names are related to user-specified mnemonic-names.

*** special names entry**. An entry in the SPECIAL-NAMES paragraph of the Environment Division which provides means for specifying the currency sign; choosing the decimal point; specifying symbolic characters; relating implementor-names to user-specified mnemonic-names; relating alphabet-names to character sets or collating sequences; and relating class-names to sets of characters.

*** special registers**. Certain compiler generated storage areas whose primary use is to store information produced in conjunction with the use of a specific COBOL feature.

*** standard data format**. The concept used in describing the characteristics of data in a COBOL Data Division under which the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.

*** statement**. A syntactically valid combination of words, literals, and separators,beginning with a verb, written in a COBOL source program.

structured programming. A technique for organizing and coding a computer program in which the program includes a hierarchy of segments, each segment having a single entry point and a single exit point. Control is passed downward through the structure without unconditional branches to higher levels of the hierarchy.

*** subject of entry**. An operand or reserved word that appears immediately following the level indicator or the level-number in a Data Division entry.

*** subprogram**. See "called program."

*** subscript**. An occurrence number represented by either an integer, a data-name optionally followed by an integer with the operator + or -, or an index-name optionally followed by an integer with the operator + or -, that identifies a particular element in a table. A subscript may be the word ALL when the subscripted identifier is used as a function argument for a function allowing a variable number of arguments.

*** subscripted data-name**. An identifier that conatins a data-name followed by one or more subscripts enclosed in parentheses.

switch-status condition. The proposition, for which a truth value can be determined, that an UPSI switch, capable of being set to an 'on' or 'off' status, has been set to a specific status.

*** symbolic-character**. A user-defined word that specifies a user-defined figurative constant.

syntax. (1) The relationship among characters or groups of characters, independent of their meanings or the manner of their interpretation and use. (2) The structure of expressions in a language. (3) The rules governing the structure of a language. (4) The relationship among symbols. (5) The rules for the construction of a statement.

*** system-name**. A COBOL word that is used to communicate with the operating environment.

T

*** table**. A set of logically consecutive items of data that are defined in the Data Division by means of the OCCURS clause.

*** table element**. A data item that belongs to the set of repeated items comprising a table.

text deck. Synonym for *object deck* or *object module*.

*** text-name**. A user-defined word that identifies library text.

*** text word**. A character or a sequence of contiguous characters between margin A and margin R in a COBOL library, source program, or in pseudo-text which is:

° A separator, except for: space; a pseudo-text delimiter; and the opening and closing delimiters for nonnumeric literals. The right parenthesis and left parenthesis characters, regardless of context within the library, source program, or pseudo-text, are always considered text words.

° A literal including, in the case of nonnumeric literals, the opening quotation mark and the closing quotation mark that bound the literal.

° Any other sequence of contiguous COBOL characters except comment lines and the word 'COPY' bounded by separators which is neither a separator nor a literal.

top-down design. The design of a computer program using a hierarchic structure in which related functions are performed at each level of the structure.

top-down development. See "structured programming."

trailer-label. (1) A file that follows the data records on a unit of recording medium. (2) Synonym for end-of-file label.

*** truth value**. The representation of the result of the evaluation of a condition in terms of one of two values: true or false.

U

*** unary operator**. A plus (+) or a minus (-) sign, that precedes a variable or a left parenthesis in an arithmetic expression and that has the effect of multiplying the expression by +1 or -1, respectively.

unit. A module of direct access, the dimensions of which are determined by IBM.

*** unsuccessful execution**. The attempted execution of a statement that does not result in the execution of all the operations specified by that statement. The unsuccessful execution of a statement does not affect any data referenced by that statement, but may affect status indicators.

UPSI switch. A program switch that performs the functions of a hardware switch. Eight are provided: UPSI-0 through UPSI-7.

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

V

*** variable**. A data item whose value may be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

*** variable length record**. A record associated with a file whose file description or sort-merge description entry permits records to contain a varying number of character positions.

*** variable occurrence data item**. A variable occurrence data item is a table element which is repeated a variable number of times. Such an item must contain an OCCURS DEPENDING ON clause in its data description entry, or be subordinate to such an item.

*** verb**. A word that expresses an action to be taken by a COBOL compiler or object program.

volume. A module of external storage. For tape devices it is a reel; for direct-access devices it is a unit.

volume switch procedures. System specific procedures executed automatically when the end of a unit or reel has been reached before end-of-file has been reached.

VSAM (Virtual Storage Access Method). A high-performance mass storage access method. Three types of data organization are available: entry sequenced files (ESDS), key sequenced data sets (KSDS), and relative record data sets (RRDS). Their COBOL equivalents are, respectively: sequential, indexed, and relative organizations.

VSE/ESA (Virtual Storage Extended/Enterprise Systems Architecture). An IBM operating system that manages multiple address spaces (partitions), up to a maximum combined virtual storage size of 256 million bytes. Address spaces of up to 2 GB are now supported (approx. 2048 megabytes).

W

*** word**. A character string of not more than 30 characters which forms a user-defined word, a system-name, a reserved word, or a function-name.

*** Working-Storage Section**. The section of the Data Division that describes working storage data items, contains either noncontiguous items or working storage records or both.

Z

zoned decimal item. See "external decimal item."

INDEX Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Y

IBM Library Server Print Preview

DOCNUM = GC26-8070-00
DATETIME = 02/17/95 14:48:12
BLDVERS = 1.2
TITLE = COBOL/VSE Migration Guide
AUTHOR =

PATH = /home/webapps/epubs/htdocs/book